DECnet/E
Network Programming in BASIC–PLUS and BASIC–PLUS–2

Order No. AA-H501B-TC

January 1982

This manual describes the DECnet/E services available to the BASIC programmer, permitting him to design and implement programs that exchange data with other programs running in a DECnet network.

OPERATION SYSTEM AND VERSION: RSTS/E V7.1

SOFTWARE VERSION: DECnet/E V2.0

To order additional copies of this document, contact your local Digital Equipment Corporation Sales Office.

digital equipment corporation · maynard, massachusetts

First Printing, January 1982

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

Distributed and Mid-Range Systems Publications typeset this manual using DIGITAL's TMS-11 Text Management System.

310ALL

Contents

Index

Figures

Tables

Preface

Manual Objectives

This manual describes the DECnet/E services available to a BASIC programmer, permitting him to code programs that exchange data with other programs running in a DECnet network. Either the BASIC-PLUS or the BASIC-PLUS-2 languages can be used.

Intended Audience

This manual is written for the experienced programmer. While the reader is not expected to be knowledgeable of network programming, he is expected to be completely familiar with both the BASIC programming language and the general mechanisms for interfacing with the RSTS/E operating system.

Prerequisite Reading

The reader is expected to be completely familiar with the concepts and features of the BASIC programming languages, as presented in the following manuals:

BASIC-PLUS Language Manual
BASIC-PLUS-2 Language Manual

The *RSTS/E BASIC-PLUS-2 User's Guide* gives instructions on how to compile BASIC programs in a RSTS/E system. The *RSTS/E Task Builder Manual* describes TKB, the linker used for building RSTS/E tasks from compiled BASIC programs.

The reader is also expected to be familiar with the concepts and features available for interfacing to the RSTS/E system monitor, as presented in the following manuals:

RSTS/E System User's Guide
RSTS/E Programming Manual

Related Documents

To obtain a general understanding of the DECnet environment — internal and external — the reader is directed to the *Introduction to DECnet.*

DECnet/E also provides utility programs that allow a terminal user to communicate with another terminal user, to manipulate files across the network, and to copy the contents of one storage device to a device on another system. The information needed to run and use these utilities is given in the companion manual, *DECnet/E Guide to User Utilities.*

Two other companion manuals tell how to generate and start a DECnet/E system (the *DECnet/E System Installation Guide*), control and manage a running DECnet/E system (the *DECnet/E System Manager's Guide*), and give other information useful to a system manager.

Three more DECnet/E manuals are available describing the DECnet/E message send/receive services for three other languages:

DECnet/E Network Programming in MACRO-11
DECnet/E Network Programming in FORTRAN
DECnet/E Network Programming in COBOL

The more advanced network programmer may want to refer to the following two manuals for information concerning various network management functions:

DIGITAL Network Architecture, Network Services Functional Specification
DIGITAL Network Architecture, Network Management Functional Specification

Structure of This Manual

This manual is divided into four main parts. Part I, "Introduction," presents a general discussion of the DECnet/E environment (Chapter 1). Part II, "Background Concepts for Network Programming," presents a tutorial discussion of the DECnet/E features available for sending and receiving messages (Chapters 2 through 4). Part III, "Network Programming in BASIC-PLUS," describes the SYS call formats used to access those features from BASIC-PLUS (Chapter 5), and Part IV, "Network Programming in BASIC-PLUS-2," describes the subroutine call formats used from BASIC-PLUS-2 (Chapter 6).

Part I
Introduction

Chapter 1
Introduction

DECnet/E is a combination of hardware and software that extends the capabilities of a RSTS/E system running on a DIGITAL PDP–11 computer to allow a programmer to develop and execute programs that exchange data with remote programs running on another DECnet system in a network.

NOTE

> Throughout this manual, the term *remote program* refers to any other program with which a program is communicating using the DECnet facilities. Such programs need not be located on another node in the network. It is possible to use DECnet/E to communicate with other programs at the local RSTS/E node. During development of network application programs, the whole system can be debugged locally and later distributed to remote nodes as required.

This manual describes the features available to network programmers in the BASIC–PLUS and BASIC–PLUS–2 programming languages on RSTS/E systems.

1.1 Basic Terms and Concepts

The term *network* is used here to refer to two or more computer systems connected such that they can exchange information. The computer systems, called *nodes*, are connected by communications paths called *physical links*. Physical links can be established through normal (switched) telephone circuits, several varieties of leased (private) lines, coaxial cables, or even satellite transmission facilities provided by several carriers.

In the network shown in Figure 1–1, a user at node BOSTON would refer to BOSTON as the *local node* and to nodes DENVER, DALLAS, LONDON, and LA as *remote nodes*. At node BOSTON, the nodes DENVER, DALLAS, and LONDON are physically *adjacent nodes*. That is, there is a direct physical link to these nodes with no intervening nodes. To a user at node LA, DENVER is the only adjacent node.

A *path* is the route over which data travels from its source to its destination within the network. *Path length* is the distance from the source node to the destination node, measured in hops. A *hop* is equal to a physical line between two adjacent nodes. In Figure 1–1, the path length between nodes DALLAS and BOSTON is one hop and between LA and LONDON is three hops. The *network diameter* is the maximum path length between any two nodes. The network shown in Figure 1–1 has a network diameter of three hops.

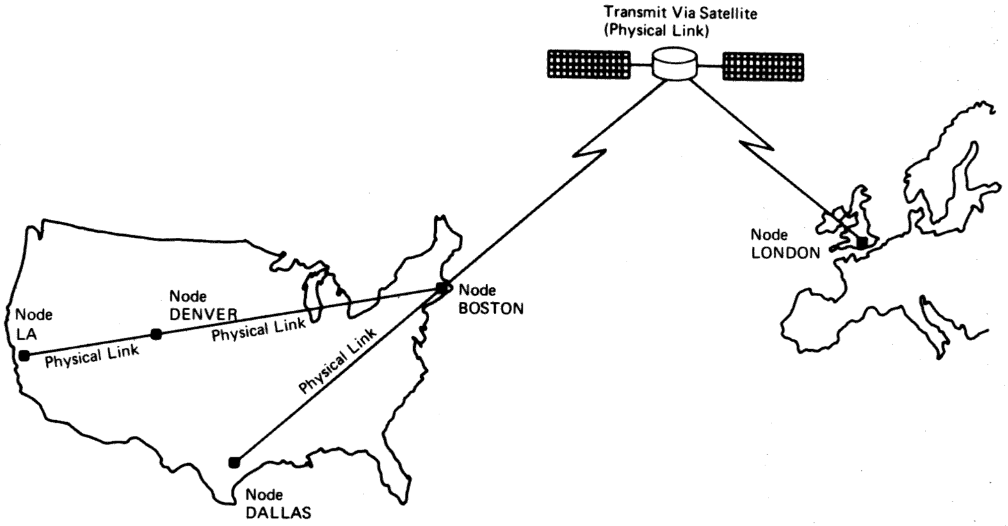


Figure 1–1: Computer Network Composed of Five Nodes

1.2 DECnet Networks

DECnet as a whole refers to the hardware and software that allows different DIGITAL systems to be connected to form networks. DECnet has been designed to allow each system to function as a separate entity but still allow access to resources that are distributed throughout the various systems in the network. A DECnet/E system provides all the capabilities of a normal RSTS/E system as well as the networking capabilities described in this manual and in the companion DECnet/E manuals listed in the Preface.

All DECnet implementations are based on a design structure called the DIGITAL Network Architecture (DNA). This architecture allows network implementations on the various DIGITAL systems to evolve (to provide more and more capability as time goes on) within a basic common structure. The structure ensures that implementations providing fewer features will always work with implementations providing more.

Perhaps a more immediate aspect of this process of evolution, however, is that at any given time capabilities can indeed differ from system to system. We do not attempt to describe here the DECnet implementations for all the different DIGITAL systems. Nevertheless, to use DECnet/E, you do need to give some thought to the other DIGITAL systems in your network.

If your network consists entirely of DECnet/E Version 2.0 nodes, this document provides all the information necessary for you to use DECnet/E to send and receive messages in BASIC programs. If your network contains other DECnet systems, however, you should check the DECnet documentation for those systems to be sure that the corresponding capabilities exist. For example, you can design network programs to be started automatically on RSTS/E systems. This feature is not available on all DECnet systems, however, so your design should not necessarily assume this capability.

A brief overview of the functional capabilities of the various DECnet implementations can be found in *Introduction to DECnet*.

1.3 The DIGITAL Network Architecture (DNA)

DECnet/E is implemented according to a set of rules, or protocols, governing the format, control, and sequencing of data exchange from the user program level down through the physical link level. These protocols are defined by the DIGITAL Network Architecture (DNA) — a logical structure that provides the model for all DECnet implementations.

DNA consists of several layers, each defining a distinct set of network functions and a set of rules for implementing those functions. Each DECnet implementation consists of software modules that perform these DNA-defined functions according to DNA-defined protocols. The relevant protocols for network programming are called the Network Services Protocol (NSP), the Transport protocol, and the Digital Data Communications Message Protocol (DDCMP).

The Network Services Protocol defines the rules for communication between programs in a network over paths called logical links. Logical links, described in detail in Chapter 2, allow many different data streams to be multiplexed onto the physical link for transmission and separated at the receiving node for delivery to the appropriate program.

The Transport protocol defines the rules for determining the actual physical path, or *route*, along which data travels to its destination.

Physical link control is achieved in DECnet by implementation of the Digital Data Communications Message Protocol (DDCMP). DDCMP ensures an error-free, sequential data path over a generally error-prone medium. This is accomplished with the Cyclic Redundancy Check (CRC) for error detection, retransmission for error corrections, and numbered data segments to ensure sequential transmission of data.

1.4 Network Routing

DECnet/E Version 2.0 is a Phase III DECnet product. (Refer to *Introduction to DECnet* for a discussion of the differences between Phase II and Phase III DECnet products.) As such, it supports the adaptive routing feature of Phase III, providing the user with the capability of communicating with other nodes in the network regardless of whether or not they are directly connected to the local node.

Networks can consist of the following three types of nodes:

- **Routing nodes** — Phase III nodes connected to multiple communications lines, supporting route-through capabilities

- **Nonrouting nodes (end nodes)** — Phase III nodes connected to a single communications line

- **Phase II nodes** — nodes running a previous generation of the DECnet architecture

Phase III nodes can communicate with and are compatible with Phase II nodes. However, Phase II nodes do not acquire any new capabilities by being connected to Phase III nodes, and the restriction that Phase II nodes can only communicate with adjacent nodes continues to apply. Thus, a Phase III node can communicate with any other Phase III node, *providing the path goes through Phase III nodes only.*

The adaptive routing feature is implemented through an algorithm that chooses the routing path with the lowest associated cost. Cost is computed as the sum of the costs of the lines over which the message is transmitted. The individual line cost parameters are assigned by the system manager and input into the system with the Network Control Program. (See the *DECnet/E System Manager's Guide* for a full discussion of line costs.) The algorithm automatically adjusts the routing when network topology or line cost changes occur.

1.5 Overview of DECnet/E Structure

The DECnet/E features for network communication are implemented as part of the RSTS/E system monitor. The manner in which these features are accessed differs with the type of user.

User programs interface with DECnet/E through NSP. Although it is not a separate entity or program, the term NSP is used throughout this manual to refer to the software that allows a user program to establish and exchange data over logical links. In other words, NSP is the implementation of the Network Services Protocol. The RSTS/E BASIC-PLUS programmer accesses NSP services directly, via SYS calls to the RSTS/E system monitor. In BASIC-PLUS-2, the programmer codes calls to MACRO-11 (assembly language) subroutines to use NSP. These subroutines can also be called from MACRO-11, FORTRAN, or COBOL programs, as described in associated DECnet/E Network Programming Manuals.

Terminal users access certain network capabilities through various DECnet/E utility programs. The utilities TLK (Talk) and LSN (Listen) allow two terminal users to type messages to each other. The utilities NFT (Network File Transfer) and FAL (File Access Listener) work together to allow network file manipulation. They are implemented according to the Data Access Protocol (DAP) so that files can be exchanged with other DECnet systems having the same capabilities. The NET utility allows the terminal user to log into and use a remote RSTS/E system as though the local terminal were directly connected to the remote system, and NETCPY copies entire devices between RSTS/E nodes. (See the *DECnet/E Guide to User Utilities* for details on these utility programs.)

The system manager uses the features provided by the Network Control Program (NCP) to monitor and control a node running in a network. (See the *DECnet/E System Manager's Guide* for details on running NCP.)

Some elements of DECnet/E are not directly accessible to any user. These lower level elements of DECnet/E include the network Transport module (TRN) that implements the Transport protocol, several hardware devices (the DMC11, DMR11, DMV11, and DMP11), and their associated software device drivers.

The device drivers are responsible for handling messages received from and sent over the physical communication lines. NSP gives a message to be sent to Transport, which selects an appropriate data path based on the destination of the message. Transport then passes the message to the proper device driver for actual transmission. The drivers manipulate the device registers to cause message transmission to occur.

To handle incoming messages, the drivers include buffer management functions that allow messages to be received from the physical links. When a message is received, it is passed to Transport, which checks the destination of the message. If the message is for the local node, Transport passes it to NSP for further processing. If the message is destined for another node in the network, Transport selects a data path and passes the message to one of the drivers for transmission.

The hardware devices are intelligent communication controllers that connect the PDP-11 to physical communications lines. The DMC11 and the DMR11 control lines that connect to only one other system in the network (point-to-point lines) while the DMV11 and the DMP11 control lines that may connect to more than one other system (multipoint lines). The physical lines can include cables, modems and telephone circuits, or even satellite transmission facilities. Each controller contains a microprocessor, its own memory, and microcode that implements the DDCMP protocol. The implementation of DDCMP in firmware considerably reduces the software overhead for the DECnet/E system.

Part II

Background Concepts for Network Programming

Chapter 2
Logical Links

This chapter introduces the logical link — the basic element of DECnet task-to-task communication. A logical link is a software path for the exchange of data between two programs running in a network.

In a network of DECnet nodes, many programs use the same physical path to exchange data. Data being sent from one program to another over a logical link is interspersed (multiplexed) with data sent over other logical links, transmitted over the physical line, and separated (demultiplexed) at the receiving end. The Network Services Protocol (NSP), implemented as a part of the DECnet software at each node, handles this multiplexing and demultiplexing. It accepts outgoing data from local programs and formats it for transmission by the communications hardware. NSP also accepts incoming data from remote nodes and separates it into individual logical link streams which are then delivered to the appropriate local programs (see Figure 2–1).

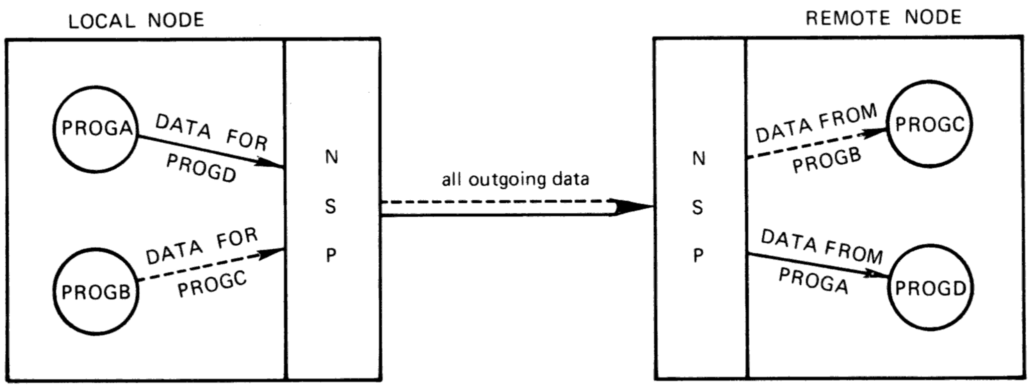


Figure 2–1: NSP Handles Multiplexing and Demultiplexing of Data over Physical Lines

Throughout the remainder of this chapter, and the two chapters that follow, various logical link parameters are discussed — object name and type code, local and remote logical link address, flow control data request counts, and so on. These parameters can be observed on an operational node by issuing the NCP SHOW LINK command (for a single link) or the SHOW ACTIVE LINKS command (for all current links). This can be very helpful when debugging a network program. (See the *DECnet/E System Manager's Guide* for details on using these NCP commands.)

2.1 Creating a Logical Link

Creating a logical link is a cooperative venture. Two programs must agree to communicate before a logical link is established. The procedure is basically the same for all implementations of DECnet. The program that initiates the request for a logical link connection is called the source. The other program is called the target. This distinction is made only during the connection sequence.

NOTE

> Other network implementations make the source program the "link master". The source must poll the target for data and only the source can disconnect the link. DECnet does not make this distinction. Once the connection is established, the terms "source" and "target" have no significance and both the local and remote programs have equal access to the logical link.

The request for a connection takes the form of a Connect Initiate Message that includes detailed information identifying the remote node and target program to which the connection is to be made. It also establishes an identifier for the link itself. In DECnet/E, the link identifier is called a user link address, or ULA. A ULA is a number from 1 to 255 that is used by the source program to refer to the link during subsequent send and receive operations.

The NSP software at the source node does some initial checking of the request (to determine if it recognizes the remote node, for example). If all is well, the source NSP forwards the connect request to the NSP at the target node. The target NSP does its own checking to determine if the connection can be made. If the connection cannot be made, the target NSP rejects the connection. Otherwise, it delivers the connect request to the target program. The target program may then choose to accept or reject the logical link.

If the target program accepts the link, it selects its own link identifier by which it will refer to the link. The two link identifiers are unrelated to one another. They can even take quite different forms if the programs are written in different languages or for different operating systems. If the remote node is a DECnet/E node, this identifier is again called a user link address and can range from 1 to 255. The number selected is completely independent of the selection made by the source program. The source and target NSPs relate the two identifiers to the same logical connection and keep the data exchanged over the link separate from data exchanged over other logical links.

The target program's confirmation or rejection of the link is passed back over the network to the source NSP, which passes this information on to the source program. This confirmation or rejection takes the form of a Connect Confirm or Connect Reject Message that is queued for the source to receive and process.

If the connection is confirmed after this mutual handshaking, the two programs can then send and receive data over the link, each referring to the link by the identifier it assigned during the connection sequence. Both programs can send data simultaneously, since the DECnet software provides logical full duplex transmission regardless of the characteristics of the intervening network. Either program can break the logical link connection when it is no longer needed.

In Figure 2–2, A shows the procedure for establishing a logical link connection. The target NSP considers the link "up" (that is, it will allow the target program to send data) when the source NSP has acknowledged receipt of the Connect Confirm Message. The source NSP considers the link up when it receives the Connect Confirm Message. In Figure 2–2, B, C, and D illustrate the three ways in which a request for a link might be rejected.

2.2 Advantages of Logical Links

In addition to making it possible to sort out the many physical streams of data that pass over a single physical line, the logical link structure maintained by NSP has several other advantages.

- Reduction of addressing data required for message transmission, thereby providing efficient line usage
- Separation of data into separate message streams
- Provision for high-priority "Interrupt" messages
- Ability to control the flow of data, thereby relieving possible network congestion

2.2.1 Efficient Line Usage

Once established, a logical link simplifies the addressing necessary for the exchange of data between two programs, both at the user's level and in transmission of data across the network. As mentioned previously, a link identifier is assigned by each program to identify an established logical link. This identifier is more convenient than the longer, detailed address used in identifying the remote node and program in the original Connect Initiate Message.

During a connection sequence, the source NSP and target NSP also establish a set of addresses that they use to identify the logical link. These addresses are called the local link address (LLA) and the remote link address (RLA), respectively. Each NSP refers to its own as the local link address and the other as the remote link address.

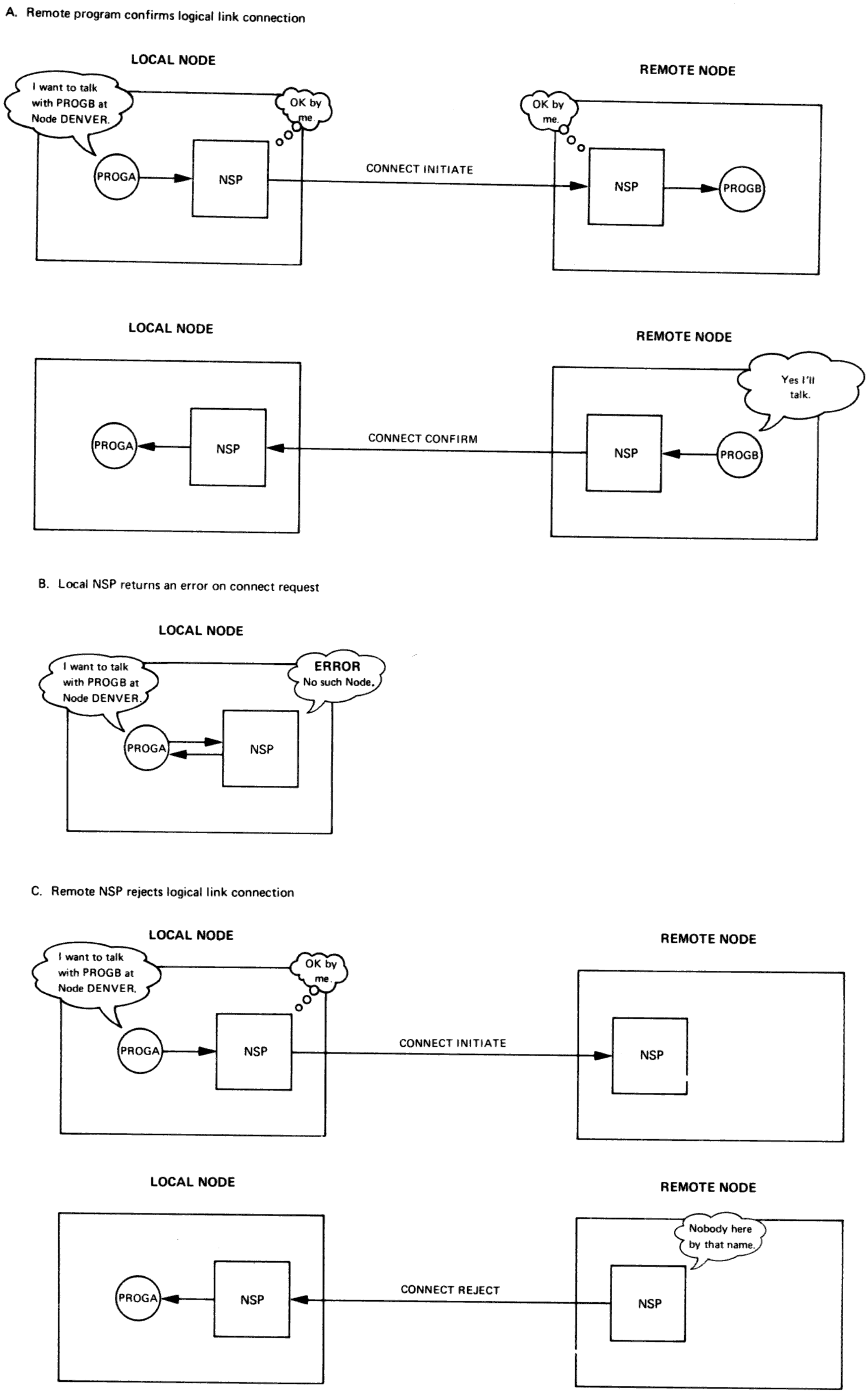


Figure 2–2: Logical Link Connections

(continued on next page)

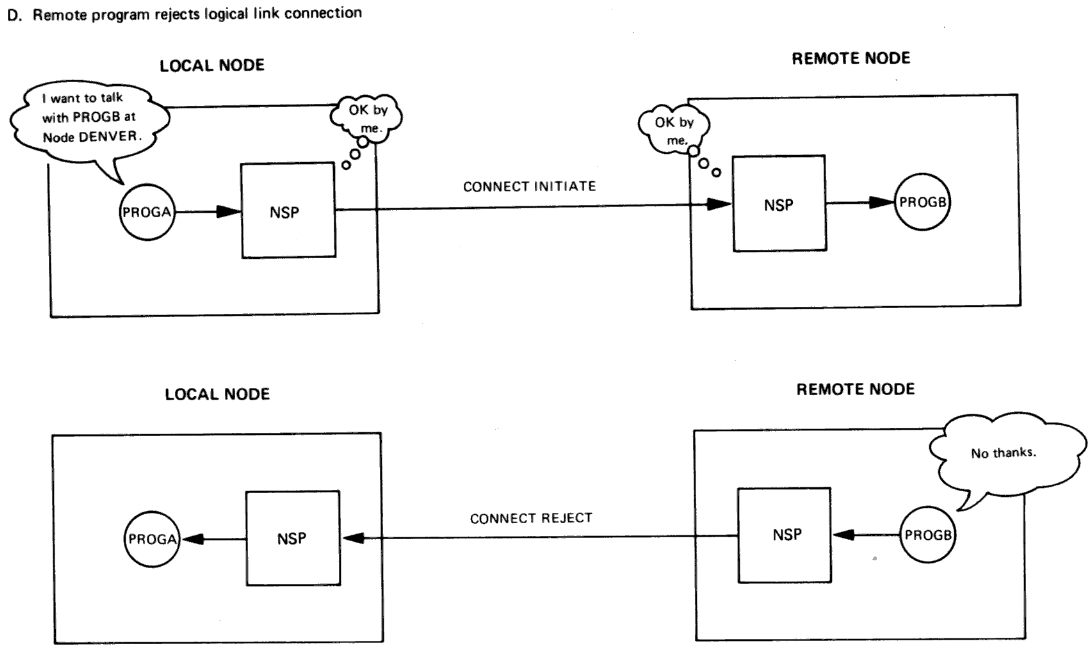


Figure 2–2 (Cont.): Logical Link Connections

While individual ULAs must be unique to the source program and to the target program, they are by no means required to be unique throughout the network, or even within the local node. Thus, the LLA and the RLA are required by NSP to separate the many distinct message streams that are multiplexed onto the communications lines. Since each NSP's LLA is unique to its own node, these NSP addresses reduce the amount of control and identifying information that must accompany the user data transferred across the link (see Figure 2–3).

Figure 2–4 illustrates how these addresses work, showing two DECnet/E nodes with two established logical links. Program A, for example, has established a logical link with Program D. Program A refers to the link with a user link address of 1. Program D refers to the link with a user link address of 23. Each NSP has established its own local link address. For Program A's NSP, the LLA is 123456. For Program D's NSP, the LLA is 024602. Each NSP keeps track of the other's LLA as a remote link address.

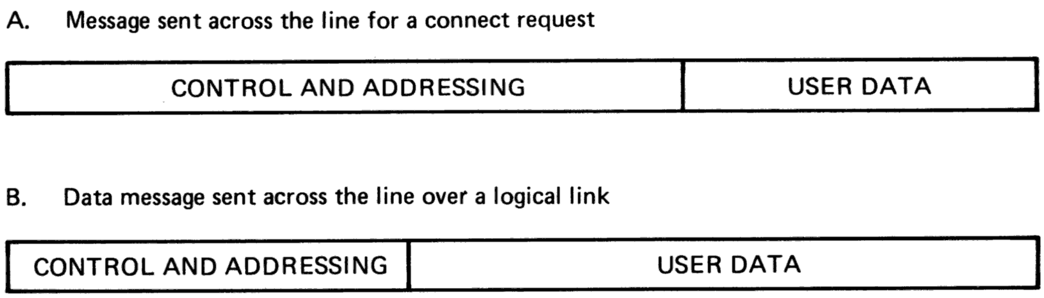


Figure 2–3: Logical Links Provide Line Efficiency

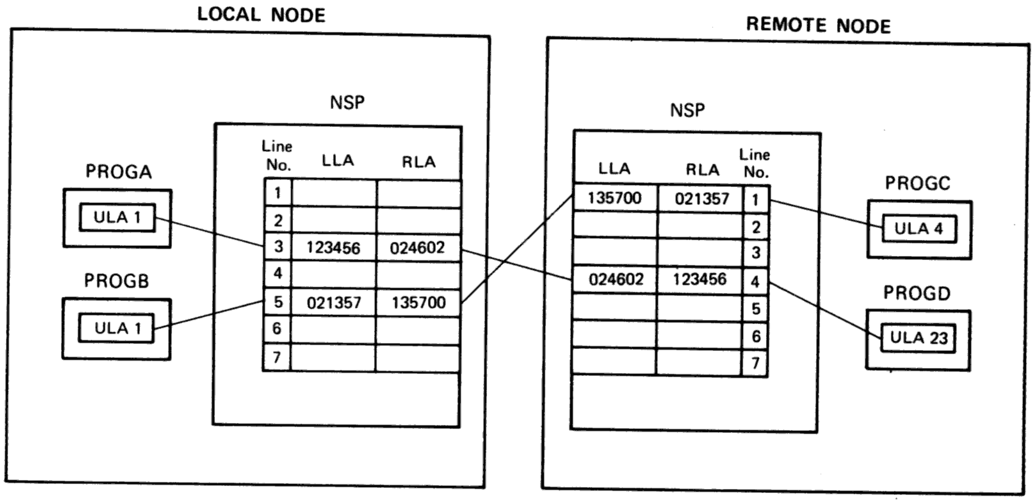


Figure 2–4: User Link Addresses, Remote Link Addresses, and Local Link Addresses for Logical Links

2.2.2 Separate Data by Intended Use

Logical links provide a means by which incoming messages can be separated into different message streams. A program can establish different logical links to communicate with different remote programs. Or it can establish several logical links with the same remote program to exchange data intended for different purposes.

Before a program can use the message services, it must first register this intention with NSP. This involves declaring the name by which the program is to be known to other remote programs in the network (that is, its identity), as well as any limits or restrictions that will affect the way in which the program can use the message services.

When a program declares its intent to use the network message services, NSP sets up a receive queue to hold pending messages. Thereafter, all messages received for the program are placed in the queue. Messages from different logical links are interspersed in the queue but the receiving program can retrieve either the first message in the queue or the first message from a particular logical link.

The concept of separate message streams for separate logical links is illustrated in Figure 2–5.

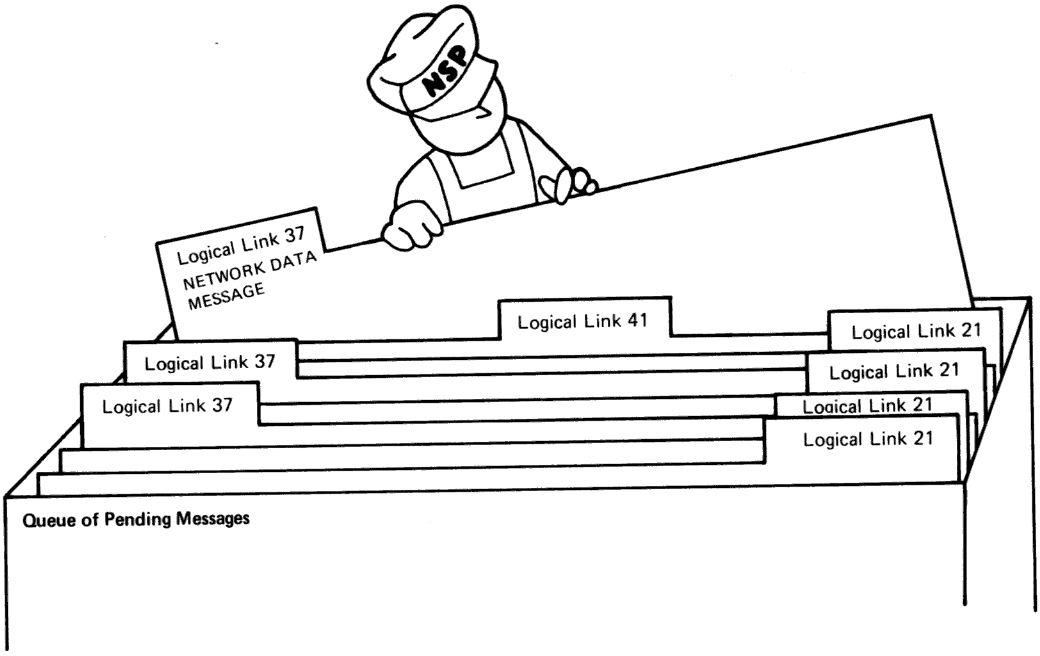


Figure 2–5: Data Is Queued for Logical Links in Separate Streams

2.2.3 Interrupt Messages

Special high-priority Interrupt Messages can be sent over logical links in all DECnet implementations. DECnet/E queues incoming Interrupt Messages at the head of the pending message queue, behind the first message in the queue (since the user program could be in the process of retrieving the first message) and behind any other pending Interrupt Messages queued for the program.

In the DECnet/E implementation, these Interrupt Messages are not true interrupts in the sense of causing an immediate jump to an interrupt processing routine. (Some DECnet implementations *do* treat Interrupt Messages as true interrupts.) They are simply messages that can be identified as different from ordinary data messages and that are placed at the head of the pending message queue (see Figure 2–6).

If a program is designed to send Interrupt Messages, the receiver at the other end of the link must be set up to process them. Interrupt Messages can contain a small amount of user data so that programs can be designed to recognize Interrupt Messages for different purposes.

Interrupt Messages are discussed in more detail in Chapters 3 and 4.

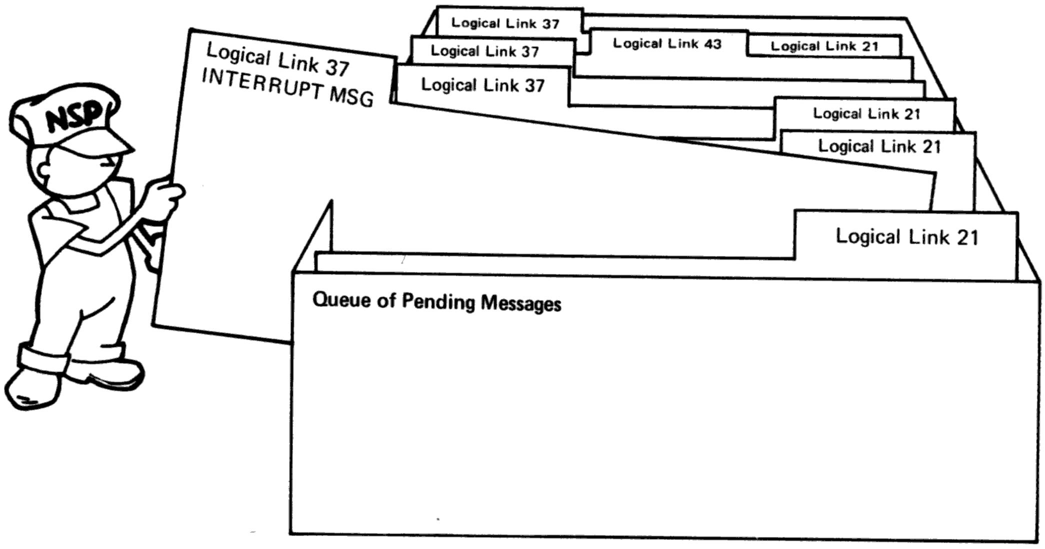


Figure 2–6: Queuing of Interrupt Messages for Logical Links

2.2.4 Flow Control

NSP maintains each program's queue of pending messages in system buffer space. A message is stored in the queue until it is received by the user program. Thus, pending messages tie up valuable system resources. Several flow control mechanisms are used in DECnet to protect systems from being flooded with messages. Some of these mechanisms are system-regulated while one mechanism is selected and controlled by the user program.

Each receiver program can select one of the following three types of program-regulated flow control:

- Segment flow control
- Message flow control
- No active flow control

A discussion of program-regulated flow control is given in Section 4.2.3.

2.3 DECnet/E Logical Link Limitations

The implementation of DECnet for RSTS/E allows a maximum of 127 active logical links for the entire node at any one time. The system manager can reduce this maximum to some number between 1 and 127. There is no limit to the number of logical links per program other than the system maximum.

Chapter 3
Overview of DECnet/E Message Services

RSTS/E provides message services for communication between local programs. The DECnet/E message services are implemented as an extension of these local services. In addition, DECnet/E offers new capabilities for local communication not found with the original RSTS/E message mechanism. The original local communication services are described in the *RSTS/E Programming Manual*. They are repeated in this document both for completeness and because they are the logical basis for the DECnet extensions.

We emphasize here that it is the way in which the user interfaces with DECnet from RSTS/E that is implemented as an extension of local capabilities. An RSX-11 programmer might interface with DECnet somewhat differently. The underlying architecture provides the consistency. The programmer at each system uses a familiar language and form. DECnet handles both the transmission itself and the translation necessary at both nodes to convert the original call to a form that can be transmitted over the line and back to the format expected at the receiving node.

Table 3-1 lists the calls used for interprogram communication by programs running under DECnet/E. The remaining sections of this chapter give a general description of each of the calls. Specific BASIC calls, with detailed formats, are given in Chapters 5 and 6.

Table 3-1: Summary of Calls for Interprogram Communication

Call	Function
Declare Receiver	Registers the program with the operating system for message services.
Remove Receiver	Terminates message operations for the program.
Get Local Node Parameters	Obtains local node information.
Log User Event	Queues a user-generated event to the local event processor for logging.
Send Local Data Message	Transmits user data to a local program.

(continued on next page)

Table 3–1 (Cont.): Summary of Calls for Interprogram Communication

Call	Function
Send Connect Initiate Message	Requests a logical link connection with another program.
Send Connect Confirm Message	Accepts a logical link connection requested by another program.
Send Connect Reject Message	Rejects a logical link connection requested by another program.
Send Network Data Message	Transmits user data to a network program over an established logical link.
Send Interrupt Message	Transmits interrupt data to a network program over an established logical link.
Send Link Service Message	Requests data over a flow controlled logical link.
Send Disconnect Message	Disconnects an established logical link, after all pending messages have been sent.
Send Link Abort Message	Disconnects an established logical link immediately, destroying any messages waiting to be sent.
Receive	Receives a message from the queue of pending messages. (One of nine message types, corresponding to the nine Send calls listed above, could be received.)

3.1 General Remarks

The Declare Receiver call informs the local RSTS/E monitor that the program issuing the call intends to communicate with other programs. This call identifies the program and defines which of the services will be required. The Remove Receiver call is issued when message operations are complete for the program and releases system resources used for message services. The Get Local Node Parameters call returns information to the calling program concerning the local node. The Log User Event call permits a user-written program to queue an event to the local event processor for logging.

Beyond these, the operations available fall into two categories: send and receive. The entities sent and received are called messages. They consist of control information identifying the type of message, and in most cases, data of interest to the programs exchanging information. Nine message types are available for these operations.

A separate call is used to send each of the message types. That is, there is a Send Connect Initiate Message call, a Send Local Data Message call, and so forth. When one of these calls is executed, NSP uses the parameters and user data included in the call to format the appropriate network message into system buffer space. Necessary header information is also included. This message is then queued to Transport for transmission across the network. The way in which a message is queued for transmission depends on the type of message. (A discussion of transmission queuing is given in Section 4.2.1.)

The Receive call retrieves a message from the program's queue of pending messages maintained by NSP in system buffer space. NSP queues incoming messages for the program as they come in across the network. Messages from local senders are also placed in the pending message queue.

NOTE

> Within the current context, *local* programs are programs running at the local node that send messages using only the Send Local Data Message call. *Network* programs, on the other hand, are programs running either locally or remotely that use DECnet to establish and send data over logical links.

The Receive call returns control information identifying the sender and the type of message (one of the nine listed previously in Table 3–1). User data accompanying the message is returned to a user buffer specified in the Receive call where it can then be examined and processed.

A user program can issue general or selective Receive calls. On a general Receive call, the first message in the queue is returned. A selective Receive call can select the first message from any local sender, a particular local sender, any network sender, or a particular logical link.

There can be times when no pending messages are in the queue when a Receive call is issued. One option available in the Receive call causes the program to "sleep", or suspend execution, until some action takes place — until a message is queued or a specified amount of time has passed, for example. The Receive call can also be issued to retrieve only part of a message. A program with limited buffer space can choose to retrieve only part of the user data with one Receive call. Subsequent Receive calls will retrieve the rest of the data, or the remaining data can be discarded.

3.2 Registering with the Monitor: Declare Receiver Call

A program declares its intent to use the message services by issuing a Declare Receiver call. This call must be executed before a program can send network messages or receive messages of any type. A program can, however, send local data messages without executing this call.

The Declare Receiver call does more than just declare intent, however. It also defines the way in which other programs throughout the network are to address this program (the program's identity) and the way in which the program can thereafter use the message services.

The Declare Receiver call has some rather far-reaching effects. For this reason, the parameters specified with the call and how the system uses them are described here in detail.

3.2.1 Declaring Identity

A Declare Receiver call identifies the calling program with an object type code, a name, or both. The manner in which a program declares its identity affects the way incoming link connections are handled.

A name is an ASCII identifier that can be used by either local or network programs to refer to the declarer. No two programs at the local node can use the same name at the same time. If a name is given, it must be unique within the local node.

An object type code, on the other hand, is simply a number in the range of 1 to 255. It can be used by other network programs to refer to the declarer. (Local programs cannot address a receiver by object code.) These codes simplify the identification of programs performing identical functions at different nodes of a network, without going through the cumbersome process of arriving at and enforcing standard naming conventions.

An object code also need not be unique, and more than one receiver can declare identity with the same object code simultaneously. A program performing a general-purpose function of interest to the entire network can incur heavy use. By permitting more than one program to declare identity with the same object code at the same time, DECnet/E permits execution of multiple copies of a program to meet sporadic heavy usage.

A range of object type codes (1 through 127) is reserved for DECnet use. The remainder (128 through 255) are available for user applications programs. Currently assigned object type codes for DECnet use are given in Appendix A.

NOTE

> In the DECnet environment it is recommended that, whenever possible, programs declare their identity with and be addressed by nonzero object type codes.

Network addressing is discussed in detail in Chapter 4.

3.2.2 Declaring Intended Usage

In addition to identifying the program, the Declare Receiver call defines how the program intends to use the message services. For example, a program can limit the incoming messages to those from local senders only. Or, it can limit incoming messages to those from network senders only.

Other protective limits can be set in the Declare Receiver call.

- The maximum amount of system buffer space to be used for the pending message queue
- The maximum number of messages that can be stored in the pending message queue
- The maximum number of active logical links that the program can have at any given time

These limits affect incoming messages and incoming requests for logical links. The Declare Receiver call does not limit the type or number of send calls a program can execute.

3.2.3 DECnet/E Use of the Declare Receiver Call

When a Declare Receiver call is issued, the system sets up a 32-byte Receiver ID Block (RIB) to associate the information with the RSTS/E job number. The Receiver ID Block actually applies to the job rather than the program that issues the Declare Receiver call. This distinction is rather fine for most applications, but does make it possible, for example, to chain together a set of programs using message services. The first program in the series would execute a Declare Receiver call. Then other programs in the chain would also be valid receivers as long as they were always executed under the same job number.

The Receiver ID Block holds the arguments passed with the Declare Receiver call, along with other system- and NSP-maintained information, such as the address pointers of the pending message queue and the address of the list of active logical links.

3.3 Remove Receiver Call

A Declare Receiver call remains in effect until a Remove Receiver call is issued. The Remove Receiver call releases the Receiver ID Block. All pending messages are discarded and all active logical links are destroyed. (NSP sends a Link Abort Message to the remote programs for any active logical links.)

Like the Declare Receiver call, the Remove Receiver call applies to the RSTS/E job. If a program does not issue a Remove Receiver call before terminating execution, other programs executed later under this job number will be unable to issue a Declare Receiver call. (As far as the system is concerned, the old one is still in effect.) A Remove Receiver call should be issued as soon as message services are no longer needed. This practice prevents queuing of unwanted messages and releases the system buffer space used for the Receiver ID Block.

Also, any program that could be stopped by a terminal user's CTRL/C should have provisions to issue a Remove Receiver call and terminate gracefully. If a job is killed, NSP automatically destroys any active logical links, discards messages, and releases any system resources used for message operations.

NOTE

> It is generally good practice for a program to issue a Remove Receiver call immediately before issuing a Declare Receiver call. This prevents errors on the Declare Receiver call caused by abnormal termination of a previous program running under the same job number.

3.4 Get Local Node Parameters Call

The Get Local Node Parameters call returns information to the calling program concerning the local node. This information consists of the node name, address, and alias (if any), and the node's default project-programmer number (if any). (See the *DECnet/E System Manager's Guide* for a discussion of alias.)

A general-purpose network program — one that runs on several different nodes rather than being specific to any one node — would use this call to obtain the node name, number, and alias for use in the connection sequence. The program uses this information to identify itself to the remote node and program. A program that is started automatically as the result of an incoming connect request would use this call to obtain the default project-programmer number for use in validating the connection and logging into a user account. (See Section 4.1.4 for a discussion of the default account.)

3.5 Log User Event Call

The Log User Event system call permits a user-written program to queue an event to the system event processor for logging. Before the event is logged, it is time-stamped by the system, in the standard Network Information and Control Exchange (NICE) protocol.

Optional parameter data can be passed to the event processor for processing and output. If it is included, this data must be in NICE protocol format. (See the *DIGITAL Network Architecture, Network Management Functional Specification*, Version 3.0.0, for further information on event logging.)

NOTE

> This system call performs a highly specialized function which requires a great deal of special knowledge of DECnet and the DIGITAL Network Architecture. It is provided as a convenience for the sophisticated network user and is not intended for normal network programs.

3.6 Send/Receive Local Data Message

The Send Local Data Message call transfers up to 532 bytes of user message data to a local program. The message is placed in the appropriate receive queue of the target program. The target program can be identified either by job number or by the name given in its Declare Receiver call.

A Receive call that returns a Local Data Message returns control information identifying it as a Local Data Message from a specific local sender, and up to 532 bytes of user data.

The Local Data Message is not used for DECnet communication but is included here for reference. A user program can carry on local conversations and maintain one or more DECnet logical links at the same time.

3.7 Send/Receive Connect Initiate Message

The Send Connect Initiate Message call is used to request a logical link connection with a remote program.

The target node and program are identified by a 120-byte Connect Data Block passed as part of the call. The following information is specified with the call:

The user link address (ULA) that the source program will use to refer to this link.

The type of receiver flow control desired by this program — that is, the type of flow control to be imposed on incoming data messages. (*See Chapter 4 for a full discussion of flow control.*)

A receive maximum. The maximum amount of data to be received by the program over this link in one Network Data Message. The size of the receive buffers allocated by NSP will also limit the amount of data that can be passed in a single Network Data Message. If the maximum buffer size set by Network Management for NSP is not large enough to handle the receive maximum as specified by the user program, NSP will alter the receive maximum to the smaller limit before forwarding the Connect Initiate Message. The local program is informed of this change (if any is made) by a field in the Connect Confirm Message from the remote program when it accepts the link.

A Connect Data Block identifying the target node and program to which the connection is requested. The target program can be identified by name or object type code. NSP inserts the local program identification, as specified in the source program's Declare Receiver call, into the Connect Data Block. This allows the remote program to determine who sent the connect request and decide whether to accept or reject the connection. The Connect Data Block can also contain data required by the remote program or system for access control. Such data might consist of a project-programmer number, password, and account number — or whatever is required by the remote program or system.

Up to 16 bytes of user data.

A Receive call that returns a Connect Initiate Message from the queue of pending messages returns control information identifying it as a Connect Initiate Message from a specific network program, and requirements imposed by the remote program. The information includes the following:

The local link address (LLA) assigned by the local NSP to refer to this logical link. The program will use the LLA to refer to the link in its responding Connect Confirm or Connect Reject Message. (Note that the LLA is only used to refer to a link while no user link address (ULA) is in effect. The ULA is established by the receiving program in its Connect Confirm Message that accepts the connect request and is used thereafter by the program to refer to the link.)

The type of receive flow control desired by the remote program — that is, the type of flow control that will be imposed on outgoing data messages over this link.

A receive maximum. The maximum amount of user data that can be received by the local program over the link. This maximum is calculated by the local NSP, based on the size of receive buffers it allocates. The program can reduce this maximum further in its responding Connect Confirm Message.

A transmit maximum. The maximum amount of user data that can be transmitted by the program over this logical link in one Network Data Message. This maximum is imposed by either the remote NSP or the remote program and corresponds to the remote program's receive maximum.

A 120-byte Connect Data Block, identifying the remote node and program requesting the link. The Connect Data Block also indicates how the remote program addressed this one (by name or object type code). If the program requires access control information from the remote program, such information would also be included as part of the Connect Data Block. The Connect Data Block and up to 16 bytes of user data are delivered to the user buffer specified in the Receive call.

3.8 Send/Receive Connect Confirm Message

The Send Connect Confirm Message call is used to accept a logical link requested by a remote program. That is, it is a positive response to a received Connect Initiate Message. The following information is specified with the call:

The user link address (ULA) by which the program will hereafter refer to this link.

The local link address (LLA) that identifies the link being accepted. (The LLA is taken from the Connect Initiate Message received for this particular link.)

The type of receive flow control desired by the calling program for incoming data messages over this link. (Flow control options are discussed in detail in Chapter 4.)

A receive maximum. The maximum amount of user data that the program wishes to receive as a unit (that is, as one Network Data Message).

Up to 16 bytes of user data.

A Receive call that returns a Connect Confirm Message from the pending message queue returns control information identifying it as such, along with requirements for messages to be transmitted over the link. The information includes the following:

The user link address (ULA) established by the receiving program in the Connect Initiate Message that requested the link. This identifies the particular link being confirmed.

The type of receive flow control requested by the remote program — that is, the type of flow control imposed on outgoing data messages. (Flow control options are discussed in detail in Chapter 4.)

A receive maximum. The maximum amount of user data that the local program will receive on this link in one Network Data Message. This will be the same or less than the size requested in the program's Connect Initiate Message. A smaller size is indicated by the local NSP if it cannot handle the size requested in the Connect Initiate Message.

A transmit maximum. The maximum amount of data the local program can send to the remote program as a unit (with one Network Data Message) over this link.

Up to 16 bytes of user data, delivered to the user buffer specified in the Receive call.

3.9 Send/Receive Connect Reject Message

A Send Connect Reject Message call is used to reject a logical link requested by a remote program. That is, it is a negative response to a received Connect Initiate Message. The following information is specified with the call:

The local link address (LLA) assigned by the local NSP to identify the link being rejected. (The local link address is taken from the received Connect Initiate Message.)

A code identifying the reason for the rejection. Ordinarily, this code is zero unless the program is rejecting the connection due to some standard reason, such as "invalid accounting data".

Up to 16 bytes of user data.

A Receive call that returns a Connect Reject Message from the queue returns control information identifying it as such and the following information:

A user link address identifying the link being rejected. This user link address will be the same as that established by the local program in the Connect Initiate Message that requested the link. The link could have been rejected by either the remote NSP or the remote program.

A code identifying the reason for the rejection if the link was rejected by the remote NSP or by the remote program for some standard reason, such as "invalid accounting data." (A list of NSP rejection codes is found in Appendix B.)

Up to 16 bytes of user data, if the link was rejected by the remote program for some nonstandard reason. This data will be delivered to the user buffer specified in the Receive call.

3.10 Send/Receive Network Data Message

The Send Network Data Message call transmits user information to a remote program over an established logical link. The logical link is identified by the user link address (ULA) established by the calling program in its Connect Initiate or Connect Confirm Message. The amount of user data that can be sent is limited to the transmit maximum established by the remote program

(see the previous discussion of a received Connect Initiate or Connect Confirm Message). Sending a Network Data Message is subject to flow control imposed by the system and by the remote program, as indicated by the flow control option selected by the remote program. A program can request that link status information be returned by a Send Network Data call and use the information returned to schedule subsequent send and receive requests. A full discussion of flow control is deferred to Chapter 4.

A Receive call that returns a Network Data Message returns identifying control information and delivers the user data to a buffer specified in the call. The amount of user data received in one Network Data Message will not exceed the receive maximum established by this program for the logical link (see the previous discussion of a sent Connect Initiate or Connect Confirm Message). Incoming Network Data Messages are subject to system flow control and to the flow control option, if any, requested by the local program in its Connect Initiate or Connect Confirm Message. If the program requests flow control, it must send Link Service Messages to request data from the remote program (see Chapter 4).

3.11 Send/Receive Interrupt Message

The Send Interrupt Message call is used to transmit high-priority data to a remote program over an established logical link. All DECnet systems maintain separate data streams for Interrupt and Data Messages. The Interrupt Message will be delivered to the remote program ahead of all Data Messages waiting to be processed at the remote end. Up to 16 bytes of user data can be sent in an Interrupt Message. Sending an Interrupt Message is subject to flow control imposed by the remote system or program. The local program can request that link status information be returned following a Send Interrupt Message call and use the information returned to schedule subsequent send and receive requests. A full discussion of flow control is given in Chapter 4.

A Receive call that returns an Interrupt Message returns identifying control information and delivers up to 16 bytes of user data to a buffer specified in the call. The Interrupt Message will have been queued at the head of the pending message queue, behind the first Data Message and behind any other pending Interrupt Messages queued for the program. Incoming Interrupt Messages are subject to flow control imposed by the local NSP. After receiving each Interrupt Message, the local program must send a Link Service Message to reenable incoming Interrupt Messages from the remote program over a logical link (see Chapter 4).

3.12 Send/Receive Link Service Message

The Send Link Service Message call is used to request data over a logical link — either to request data from a remote program over a flow controlled link, or to reenable incoming Interrupt Messages from the remote program. In these two cases, detailed link status information can be returned or the call can be issued solely to obtain this information.

A Link Service Message is returned on a Receive call when a link becomes unblocked. If a local program has tried to send a Network Data or Interrupt Message and failed because the remote system or program has inhibited flow, the local NSP informs the local program when the condition clears. To encourage efficient processing of pending messages, NSP returns a Link Service Message on a Receive call only when the pending message queue is empty. Detailed link status information is returned that the local program can use in regulating its send and receive requests. (The format of the status information for a received Link Service Message is the same as that returned, upon request, for a Send Network Data, Interrupt, or Link Service call.)

For a more complete understanding of the uses of Link Service Messages, see the discussion of flow control in Chapter 4.

3.13 Send/Receive Disconnect Message

The Send Disconnect Message call is used to break an established logical link provided that all Network Data, Interrupt, and Link Service Messages previously sent over the link have been acknowledged by the remote NSP. If they have not been acknowledged, the call will terminate with an error and the link will not be broken. Successful completion of the call means that no more messages can be sent over the link. The user link address is freed and can be used to establish another logical link. However, messages already in the pending message queue will remain. They can (and should) be cleared from the queue by doing selective Receive calls using the user link address of the disconnected link. Once the link has been disconnected, new messages received for the link will be discarded by NSP. The Disconnect Message is sent to the remote system. When it is received by the remote NSP, the link is broken at that end. Up to 16 bytes of user data can be sent with a Disconnect Message.

A Receive call that returns a Disconnect Message from the pending message queue returns control information identifying it as such, as well as a user link address and up to 16 bytes of user data. Receipt of a Disconnect Message indicates that the remote program has terminated the logical link. Furthermore, it indicates that all ~~previous~~ Network Data Messages, Interrupt Messages, and Link Service Messages sent by the remote program were placed in the pending message queue by the local NSP before the Disconnect Message was received. Once the Disconnect Message is received by the local NSP, the local program cannot send any more messages over the link.

The Disconnect Message is useful in "master-slave" communication when the master program only transmits data and the slave program only receives data. Since there is no two-way communication, the master program can issue a Disconnect Message to break the link with the assurance that the remote system has received all the data sent and queued it for the slave program. Successfully completing the disconnect does not guarantee that the slave has processed the data, however. Where such guarantees are desirable, or where two-way communication is desired, programs should agree to terminate the logical link by exchanging Network Data Messages, Interrupt Messages, or whatever.

3.14 Send/Receive Link Abort Message

A Send Link Abort Message is used to break a logical link immediately. Unlike a disconnect, no attempt is made to ensure that messages previously sent over the link have been acknowledged by the remote system. In fact, NSP discards any messages it has queued for transmission for the link. (Section 4.2.1 describes NSP transmit queuing.) No further data can be sent over this logical link. Received messages already in the pending message queue will remain but new messages received after the Link Abort Message is issued will not be queued for the link. Messages already in the queue can (and should) be cleared by doing selective Receive calls using the user link address of the aborted link.

If the link is established, the Link Abort Message is sent to the remote system as notification that the link is broken and up to 16 bytes of user data can accompany the Link Abort Message. A Link Abort Message can also be sent for a logical link that has not yet been confirmed by the remote program. In this case, NSP breaks the link at the local node, but it does not notify the remote system. (Hence, the user data does not reach the remote program.) Should the remote program confirm the link, the local NSP no longer knows the link and it so informs the remote NSP, which notifies the remote program. Should the remote program reject the link, the local NSP informs the remote NSP that it has no such link but since the remote program rejected the link anyway, no notification is given to the remote program.

A Receive call that returns a Link Abort Message from the pending message queue returns control information identifying it as such, as well as a user link address and a local link address to identify the link aborted. The local link address identifies a link aborted during the initial connection sequence. For example, if the local NSP received a Connect Initiate Message, but the local program has not yet responded with a Connect Confirm or Connect Reject Message, a Link Abort Message is queued with the same local link address as was specified in the received Connect Initiate Message. The link could have been aborted by the remote NSP or the remote program. If the link was aborted by the remote NSP, the message will contain a reason code identifying the reason for the rejection. If the link was aborted by the remote program, up to 16 bytes of user data can also be delivered with the Link Abort Message. Once the local NSP receives the Link Abort Message, the local program cannot send any more messages over the link.

Chapter 4
Network Addressing and Flow Control

With the background given in Chapters 2 and 3, we can now discuss two areas of interprogram communication in a DECnet network: network addressing and flow control. Network addressing concerns the methods whereby a program declares its identity for the purpose of receiving network messages. Flow control involves the various techniques used by DECnet to ensure an orderly transfer of message and control data between nodes in a network.

4.1 Network Addressing

When a program declares its intent to use the network message services, it is influenced by the following DECnet/E features:

- The options available in the Declare Receiver call
- The way in which DECnet/E NSP handles incoming requests for logical link connections
- The multiple copy execution feature, by which more than one copy of a network program can be executed in the RSTS/E timesharing environment
- The automatic job startup feature, by which DECnet/E allows programs to be defined by the system manager for automatic startup on receiving a request for a logical link connection

4.1.1 Declaring Network Names and Object Type Codes

Every DECnet/E program that wishes to receive messages from local senders or establish logical links for data exchange with other network programs must declare its identity in a Declare Receiver call. The program can declare its identity with a nonzero object type and with a null name — abbreviated to DR(n,null) in this chapter — with a zero object type and a nonnull name — DR(0,name) — or with a nonzero object type code and a nonnull name — DR(n,name). Remember from the discussion in Chapter 3 that nonzero object type codes should be used consistently throughout the network.

Declaring identity by object type code alone — DR(n,null) — permits multiple copies of a program to be executed without the necessity of generating a unique identity for each copy. DECnet/E allows more than one program to run and declare identity with the same object type code. Thus, several copies of a program can be executed simultaneously in response to incoming connect requests.

On the other hand, declaring identity by name alone — DR(0,name) — requires uniqueness in the sense that no other program at the local node can declare identity with the same name at the same time. If a program declares its identity with the same name each time it runs, multiple copies cannot execute simultaneously.

It is possible, however, to combine the features of uniqueness and of multiple copy execution. By appending the RSTS/E job number to a common root, a program can create a unique, nonnull name — for example, FAL21, FAL05. (The RSTS/E job number can be determined with the Return Job Status system call, described in the *RSTS/E Programming Manual*.) The program can then declare its identity with both the name created *and* a nonzero object type code. This permits multiple copies of the program to be run as a result of incoming connect requests but permits each copy to create its own unique identity.

4.1.2 Handling of Incoming Connect Requests

When a Connect Initiate Message is received at the local node, NSP searches its list of Receiver ID Blocks for the proper receiver. (All other incoming messages reference an established logical link and will be queued to the correct program without searching the list of Receiver ID Blocks.) The method of addressing specified in the Connect Initiate Message determines how NSP processes the request.

Incoming requests for logical link connections can be addressed to either an object type code alone — CI(n,null) — or to a name alone — CI(0,name). DECnet/E does not allow incoming connect requests to be addressed using both object type code and name. It will reject such requests, sending back a Connect Reject Message with a reason code indicating that the requested program cannot be found.

Incoming Connect Request Addressed to Object Type Code — CI(n,null)

If the incoming Connect Initiate Message is addressed to an object type code — CI(n,null) — NSP checks its Receiver ID Blocks to see if one or more programs are running that have declared identity with this object type code. The Declare Receiver call could have specified either DR(n,null) or DR(n,name).

If one or more such receivers are found that have not reached their declared logical link maximum or message maximum, NSP queues the connect request for the first such receiver in its list (ordered by job number).

NOTE

> Remember from Chapter 3 that a program sets its own limits on the number of logical links allowed and the size of the pending message queue in the Declare Receiver call.

If there is no such receiver or if there are one or more such receivers but all have reached their declared logical link maximum or message maximum, NSP attempts to start another copy of the program. This process is described fully in Section 4.1.4 and also in the *DECnet/E System Manager's Guide*. Briefly, NSP checks to see if the program identified in the Connect Initiate Message has been defined by the system manager for automatic startup. If so, NSP creates a job and starts the program. The program must then issue a Declare Receiver call with the appropriate object type code. This creates a Receiver ID Block and a pending message queue to which the waiting Connect Initiate Message can then be transferred. If the program does not issue a Declare Receiver call within two minutes, NSP rejects the link by returning a Connect Reject Message to the remote program.

Incoming Connect Request Addressed to Name — CI(0,name)

When the incoming Connect Initiate Message is addressed to a name — CI(0,name) — NSP checks its list of Receiver ID Blocks to see if a program is running that declared its identity with this name. The Declare Receiver call could have specified either DR(0,name) or DR(n,name).

If such a receiver program is running, and it has not yet reached its declared logical link maximum or message maximum, the Connect Initiate Message is placed in the receiver's queue of pending messages.

If no such program is running, NSP will attempt to start the program. If it is successful in doing so, the program started must declare its identity with the appropriate name. Again, if the program fails to issue a Declare Receiver call within two minutes, NSP rejects the connection.

If such a program is running, but it has reached its declared logical link maximum or message maximum, NSP rejects the connection, returning a Connect Reject Message with an appropriate reason code explaining the rejection. Automatic startup is not attempted in this case, because a program is already running with the name to which the Connect Initiate Message was addressed. A second copy with the same name cannot be generated since receiver names must be unique.

4.1.3 Multiple Copies

In a DECnet/E system, multiple copies of a program can be started by a local terminal user, through the RSTS/E BATCH processor, or through the automatic job startup feature. The manner in which copies of a program can be started has some bearing on the way in which a Declare Receiver call should be issued.

Multiple Copies Initiated by Terminal Users

A program run from a local terminal with the purpose of communicating with a remote program will normally send at least the initial Connect Initiate Message requesting a logical link. (It is unlikely that remote users would wish to have the success of incoming link connections rely upon someone starting the program manually.)

In this case, the program can declare its identity with DR(n,null), or with DR(n,name) or DR(0,name) as long as the program generates a unique name with each copy, as described in Section 4.1.1. The choice depends on whether or not the program receives any connect requests, and if so, if it is necessary for the remote program to connect with a specific copy.

Suppose, for example, that the local program issues the initial Connect Initiate Message for establishing the first logical link. Suppose further that it is then necessary for the remote program to establish a second logical link with the same copy that sent the Connect Initiate Message for the first link. The local program can declare its identity with either the form DR(0,name) or DR(n,name). The unique copy name is passed to the remote program in the Connect Data Block in the Connect Initiate Message. The remote program can then address its connect request in the form CI(0,name) to ensure that the request is delivered to the same copy that issued the first Connect Initiate Message.

Multiple Copies Executed under BATCH

Multiple copies of a program can also be executed through use of the RSTS/E BATCH processor (see the *RSTS/E System User's Guide*). Again, such programs will normally be responsible for initiating at least the first logical link connection. The techniques described previously for multiple copies started by terminal users would also apply here.

Multiple Copies Initiated through Automatic Startup

Multiple copies of a program can also be initiated for programs that the system manager has designated for automatic job startup. Such copies result from incoming connection requests addressed CI(n,null). (See Section 4.1.4 for full details.) The local program can declare its identity with DR(n,null), or with DR(n,name) as long as the name is unique for each copy.

4.1.4 Automatic Job Startup

Using the Network Control Program (NCP), the system manager can install user programs for automatic startup. Such programs reside as files on disk in the normal fashion and can be started by NSP to fulfill incoming requests for logical link connections (see Section 4.1.2). This installation method is described fully in the *DECnet/E System Manager's Guide*. Briefly, a DEFINE OBJECT or SET OBJECT command in NCP relates the file name to, again, an object type code, a name, or both. These commands also allow two parameters to be associated with a program. For BASIC programs, the first parameter is used as a starting line number. The second parameter is made available to the program when it is automatically started.

NSP maintains a list of these definitions in its parameter file. The definitions are ordered by object type code, and thus, there can be only one program defined for automatic startup for each object type code. In particular, there can be only one program defined for automatic startup with an object type code of zero — (0,name). The same program can, however, be defined more than once, with different names, object type codes, starting line numbers, and

parameters. In other words, it is possible to design a program with different entry points when it is addressed by different names or object type codes in the incoming connect requests.

When a program must be started automatically, NSP checks the object type code or name given in the incoming connect request against its list of defined programs. If a match is found, NSP executes a monitor request to run the program as a detached job under RSTS/E.

Once started, the program must declare its identity with the object type code or name that was specified in the Connect Initiate Message that caused the program to be started. If it does not, the connect request will never be delivered to the proper pending message queue.

NSP passes this information to the started program in core common. Any optional parameter established by the DEFINE OBJECT or SET OBJECT command is also passed in the string. A BASIC program can retrieve the core common data by issuing the Get Core Common String system call (SYS call 7), as described in the *RSTS/E Programming Manual*. The format of the core common string is as follows:

Byte(s)	Contents
CORCMN+0	Length of data in core common
+1	Object type code (0–255)
+2-7	Name (ASCII)
+8-9	Second DEFINE parameter (-32768 through 32767)
+10	Reserved

The mnemonic CORCMN is assigned by the file COMMON.MAC to octal location 460 (see the *RSTS/E System Directives Manual*).

If the object type code in byte 1 of core common is nonzero, the program must declare its identity with that object type code. It can also use a nonnull unique name, if desired. If the object type code is zero, the program must declare its identity with the name given in bytes 2-7 of core common. It can also declare its identity with a nonzero object type, if desired.

When the Declare Receiver call is issued, NSP transfers the waiting connect request to the program's queue of pending messages as a Connect Initiate Message. NSP waits two minutes from the time it starts the program for the connect request to be transferred.

If the connect request is not transferred to a queue within this time period, NSP rejects the connection, sending a Connect Reject Message to the remote program. Since the remote program could also abort the link within this period, the first action of an automatically started program (after issuing the Declare Receiver call) should be to issue a Receive call to check for pending messages. If there are none, the program should issue a Remove Receiver call, and terminate execution with the Kill Job system call. (See the *RSTS/E Programming Manual*.) Once the connect request has been received, the pro-

gram should respond to the request immediately. While the local NSP's timer is stopped when the message is transferred to the receiver's pending message queue, the remote node could be executing a timeout on the Connect Initiate Message.

A program that has been started in response to an incoming Connect Initiate Message runs as a privileged RSTS/E user, and it is the responsibility of that program to protect the rest of the system from unauthorized outside access. Thus, after receiving the Connect Initiate Message, the program should check the accounting information passed in the Connect Data Block to determine the user account (and, therefore, the privilege level) in which it should run. If no such information is specified in the Connect Data Block, the program should issue a Get Local Node Parameters call to obtain the local node's default project-programmer number, if any. If a default account has been specified, the program should then look up the password for that account using the Read Accounting Data system call and log into the default account. (See the *RSTS/E Programming Manual.*) If the local node has no default account, the connect request should be rejected. (Programs that always log into a predetermined account need not go through this procedure.)

4.1.5 Examples of Network Addressing

This section presents three examples of network addressing using the features just discussed.

The first example (Figure 4–1) shows the network addressing used by two DECnet/E utility routines — the Network File Transfer utility (NFT) and the File Access Listener (FAL). NFT is run by local users to connect with a remote FAL to access files at remote nodes. FAL is run at the local node as result of incoming connect requests from a remote NFT. Both programs are designed to execute multiple copies as demand requires.

Since NFT is started only by local terminal users and never by remote request, an object type code is not necessary. NFT is loaded from the system disk and declares its identity with an object type code of zero and a unique name each time it is run. NFT issues the Connect Initiate Message to establish a logical link for the transfer of data to or from remote nodes as specified by the user.

Copies of FAL are started by incoming Connect Initiate Messages addressed to object type code 17. FAL declares its identity with object type code 17 and a unique name. It also declares that the maximum number of logical links allowed is one so each incoming connect request starts a new copy.

The second example (Figure 4–2) shows NETCPY, a device copying program that can be started either by local terminal users or by incoming connect requests. The program copies an entire device between nodes. Multiple copies

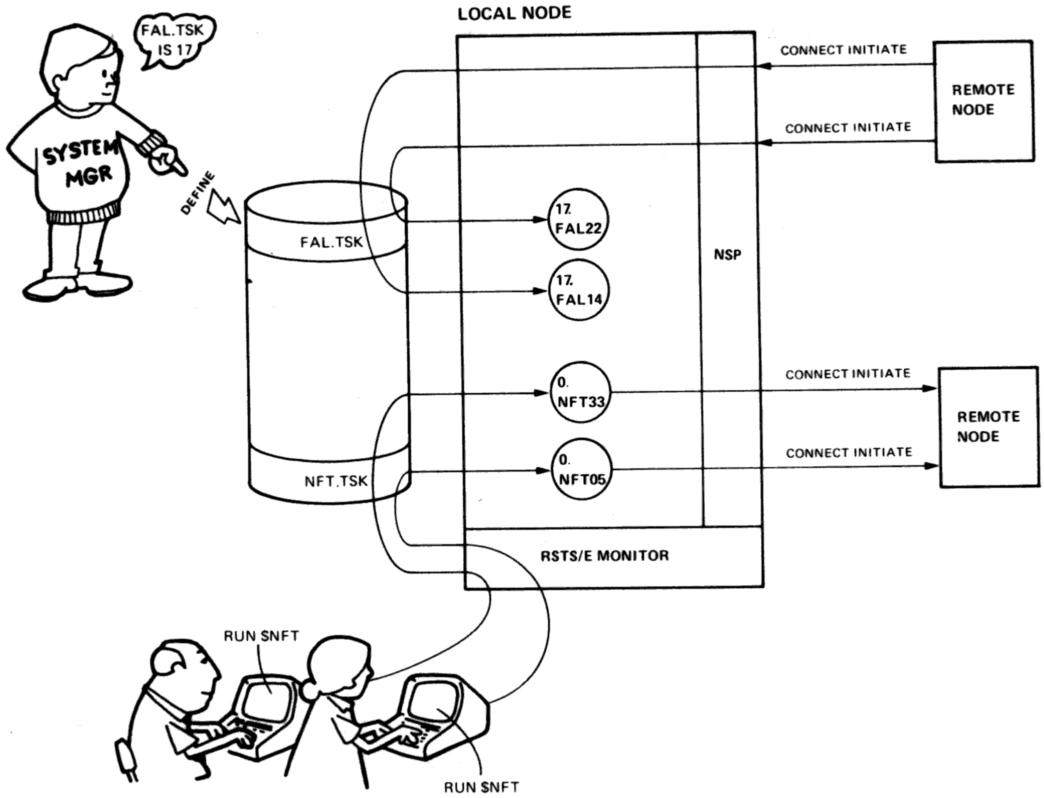


Figure 4–1: Network Addressing Used by NFT and FAL

are executed according to demand. Here, the Declare Receiver call is different depending on how the program is started. NETCPY is a BASIC–PLUS program and takes advantage of the starting line number feature. Using the Network Command Program (NCP), the system manager issues a command to SET or DEFINE NETCPY.BAC as object 20, with a starting line number of 29000. If NETCPY is started by a Connect Initiate Message from a remote program addressed to object type code 20, the program declares its identity with object type code 20, a unique name, and a link maximum of one. If it is started by a local terminal user, it declares its identity with an object type code of 0, a unique name, and a link maximum of zero. Declaring a zero link maximum ensures that a program started by a local user will never have a Connect Initiate Message queued from a remote program.

The third example (Figure 4–3) shows a program that can be run either by a local terminal user or as a result of an incoming connect request. The local terminal user is presented with a data display. Parameters are selected by the user and passed to a remote program monitoring a data gathering device. At the end of the day, the remote program passes the data back to the local program for analysis and to the local terminal user, who repeats the process.

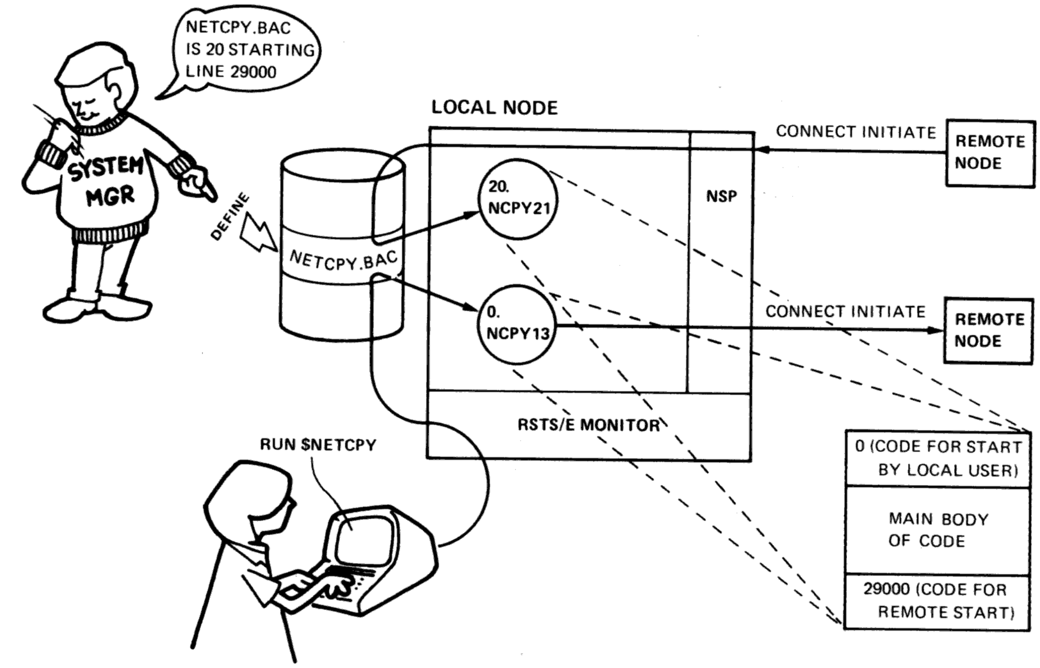


Figure 4–2: Network Addressing Example — Different Declare Receiver for Local and Remote Start

Since only one copy of the program is ever needed at any given time, it declares its identity with a zero object type code and the name STAT. It is stored as file RSTAT.SAV on disk, where it can be run by the local user. Using NCP, the system manager has issued a command to SET or DEFINE RSTAT.SAV as STAT.

4.2 Flow Control

DECnet flow control protects user programs as well as entire systems from being flooded with more messages than can be processed at any given time.

A DECnet/E sender is subject to flow control regulations imposed by the local NSP, the remote NSP, and the remote program (if the remote program requested flow control when the link was established). As a receiver, a DECnet/E program can select from three flow control options for a logical link to regulate the amount of data sent to it by the remote program.

As a sender and/or receiver, a program must deal with the following four aspects of DECnet/E operation:

- **Transmit queue management**, imposed by the local (sending) NSP, ensures that a local program does not use too much system buffer space for transmission.

- **Backpressure flow control**, imposed by the remote NSP, is used as a safeguard against excessive incoming data messages.

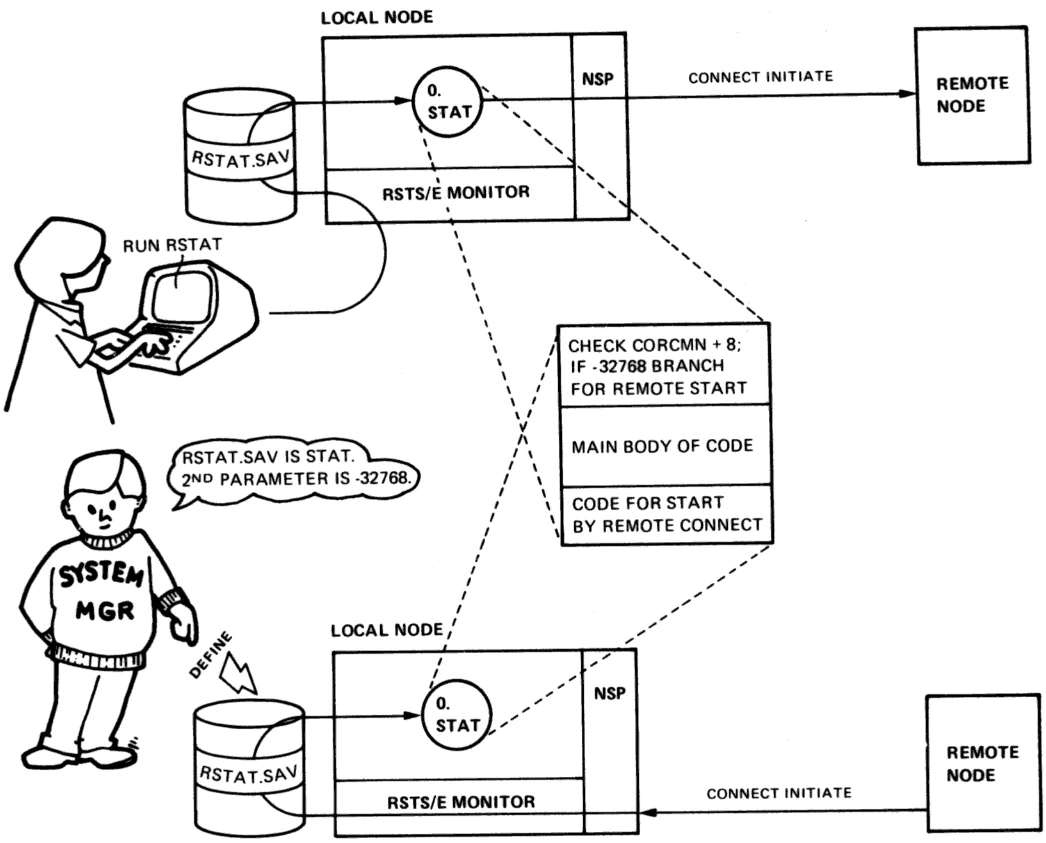


Figure 4–3: Network Addressing Example — Declaring Identity by Name Alone

- **Data message flow control**, imposed by receiving user programs, enables a program to control the flow of data to it. Both programs using a logical link can (independently) choose to receive network messages only when they specifically request data from the other program; or each can choose no flow control, placing no restrictions on the amount of data that they will receive.

- **Interrupt message flow control**, imposed by either the remote NSP or the remote program (depending on the DECnet implementation), protects against Interrupt Message congestion. A DECnet/E program can receive only one unsolicited Interrupt Message over a link. After that, the program must request an Interrupt Message before the remote program can send one.

4.2.1 Transmit Queue Management

Once a logical link is established, DECnet/E maintains two transmit queues for the link: a transmit queue for Network Data Messages and a transmit queue for Interrupt and Link Service Messages. Both queues hold messages waiting for acknowledgment from the remote NSP.

For example, when a DECnet/E program sends a Network Data Message to a remote program over a logical link, the message is queued to the Transport

software for transmission and is also placed in the data transmit queue. The Network Data Message is held there until the local NSP receives an acknowledgment from the remote NSP indicating that it has received the message.

If NSP does not receive an acknowledgment within a reasonable period of time, it assumes that the message has been lost and requeues it to Transport for retransmission. If the message has not been successfully transmitted after a specific number of retries, NSP aborts the link with a reason code indicating that the remote system is not responding.

The timeout period used depends on the message type. For outgoing Connect Initiate Messages, a predetermined value is used. Once a logical link is established, however, network management dynamically computes the timeout period, using various network parameters defined by the system manager using NCP and the SET EXECUTOR command. (See the *DECnet/E System Manager's Guide* for details.)

This acknowledgment procedure between the two NSPs ensures that messages sent over the logical link have been received, and that they were received in the same order they were sent.

Each of the transmit queues for a logical link can hold up to eight messages. (This limit can be changed by the system manager to any number from one to eight to further protect system buffer space.) When one of the transmit queues for a logical link is filled with messages waiting for acknowledgment, further transmission of that type of message over the logical link is inhibited by the local NSP until one or more of the queued messages have been acknowledged by the remote NSP.

An error is returned to the program if it issues a send request while the transmit queue for that message type is full. This condition is temporary. When the condition clears, send requests are again allowed for that message type. A program getting such an error on a send request should simply wait a short time (1 to 5 seconds) and retry the send request.

The state of the transmit queues also affects the sending of Disconnect Messages. If any messages are waiting for acknowledgment in the Data or Interrupt/Link Service transmit queue, a Send Disconnect Message call for that logical link will not be allowed and will result in an error. Thus, a program sending a successful Disconnect Message can assume that all previously sent messages have been received and acknowledged by the remote NSP. A Link Abort Message for that logical link will be allowed, however, and any messages waiting in the transmit queues will be discarded.

Figure 4-4 illustrates transmit queue flow control. PROGA has established two logical links, and PROGB and PROGC have each established one logical link. Each logical link has its own Data and Interrupt/Link Service Message transmit queues. PROGA issues a Send Network Data Message system call. The message goes to the Data transmit queue for that logical link and is also queued for transmission by the communication hardware. At the device driver level, all network messages (both data and control messages) await transmission over the physical line.

PROGB is shown trying to send a Network Interrupt Message. The Interrupt/Link Service transmit queue for that link is full, however, so NSP returns an error on the send. PROGC issues a Disconnect Message for its logical link.

Since messages are still waiting for acknowledgment in the transmit queues, the Disconnect Message is not allowed and an error is returned. (Sending a Link Abort Message would have broken the link but the messages would have been discarded.)

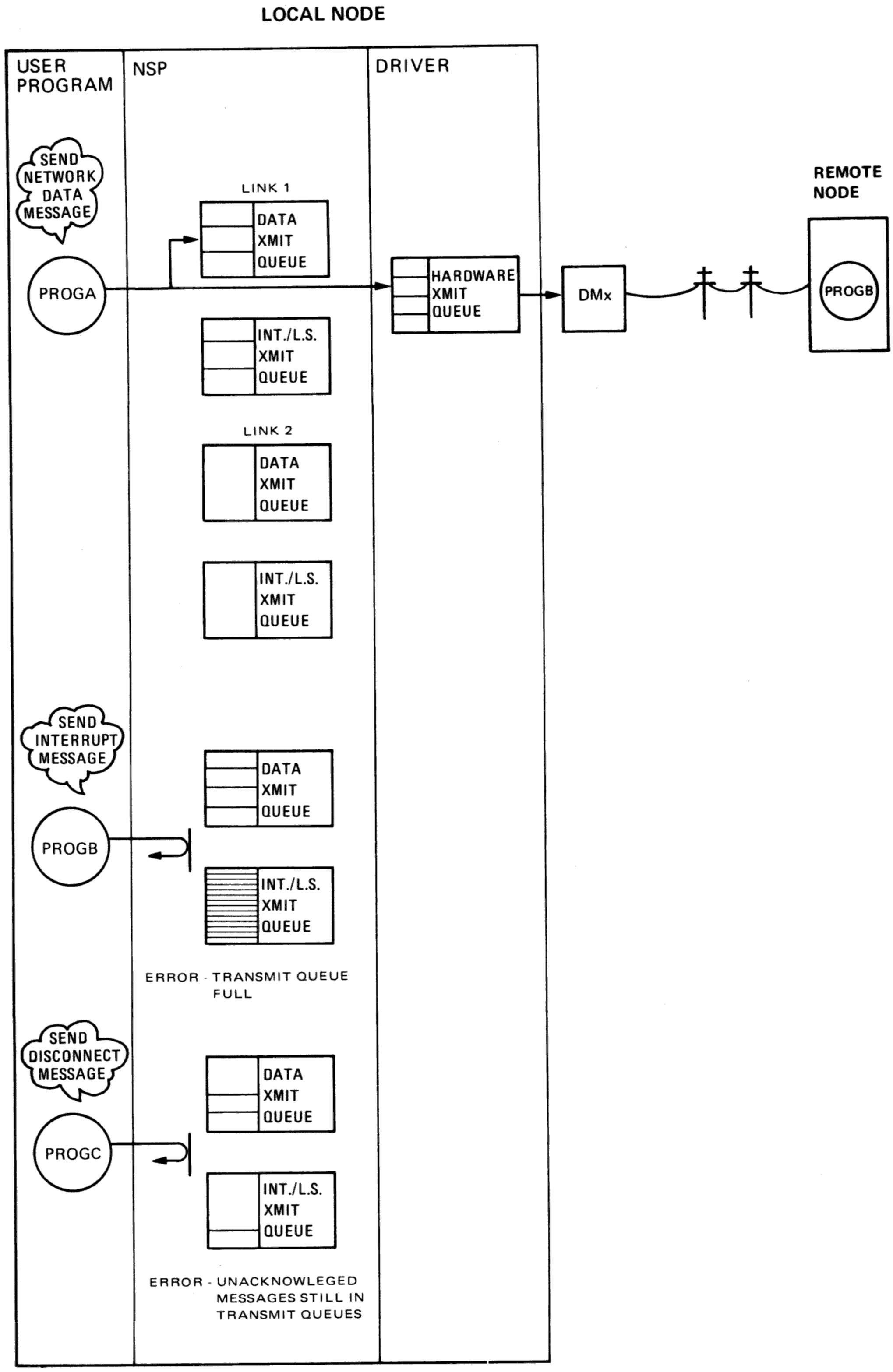


Figure 4–4: How Transmit Queue Flow Control Affects a DECnet/E Program

4.2.2 Backpressure Flow Control

NSP at the remote node can inhibit the local program from sending Network Data Messages over a logical link. The reasons for this blocking can vary from system to system: lack of system buffer space, system load, or some similar condition. This type of flow control, whereby the *remote* NSP can inhibit message flow from the local program, is known as backpressure.

When the remote NSP receives a Network Data Message that it cannot process, it returns a negative acknowledgment (NAK) for the message, along with an error code informing the local NSP that data message flow to the remote program is being inhibited on the link over which the message was sent. The local NSP then inhibits transmission of Network Data Messages over the indicated logical link until further notice is received from the remote NSP. (Note that the local program can still receive messages over the logical link.) When the remote NSP informs the local NSP that the backpressure condition has cleared, the local NSP retransmits the message that was previously-NAKed and reenables transmission of Network Data Messages over the logical link.

If the backpressure condition clears before the local program attempts another send over the link, the program is not informed that the conditon occurred. However, if the program attempts to send another Network Data Message while transmission on the link is still inhibited, the local NSP returns an error to the program (see A, in Figure 4–5). When the condition clears, the local NSP delivers a Link Service Message to the program, informing it that transmission of Network Data Messages is again allowed for the indicated logical link (see B, in Figure 4–5). This Link Service Message is delivered to the local program on a Receive call when the queue of pending messages is empty.

Note that the remote NSP enforces flow control for specific logical links, not for the entire node.

From the reverse viewpoint, the local NSP will inhibit incoming data messages over a logical link when a Network Data Message is received from a remote program over the link and the pending message queue of the intended receiver is full. (Remember that the local program itself limits the size of its pending message queue in its Declare Receiver call.) The local NSP informs the remote NSP that incoming data message flow has been inhibited by sending a message to the remote system.

Under these conditions, the Network Data Message is not queued for the local program, but is NAKed by the local NSP. This forces the remote NSP to retransmit the message after flow is reenabled. Any data messages already transmitted by the remote node will also be NAKed by the local NSP.

Furthermore, the local NSP marks all other logical links on the Receiver ID Block to be turned off if any Network Data Messages are received on those links. However, no message is sent to the remote system unless a Network

Data Message is received on one of these links before the restriction is lifted. (Note that while the local NSP will no longer accept incoming messages, the local program can still transmit.)

When the local program empties its pending message queue, any links that were actually turned off are turned back on and a message is sent to the remote NSP to reenable transmission. Those links that were merely flagged to be turned off are restored to normal operation, but no message is sent to the remote system.

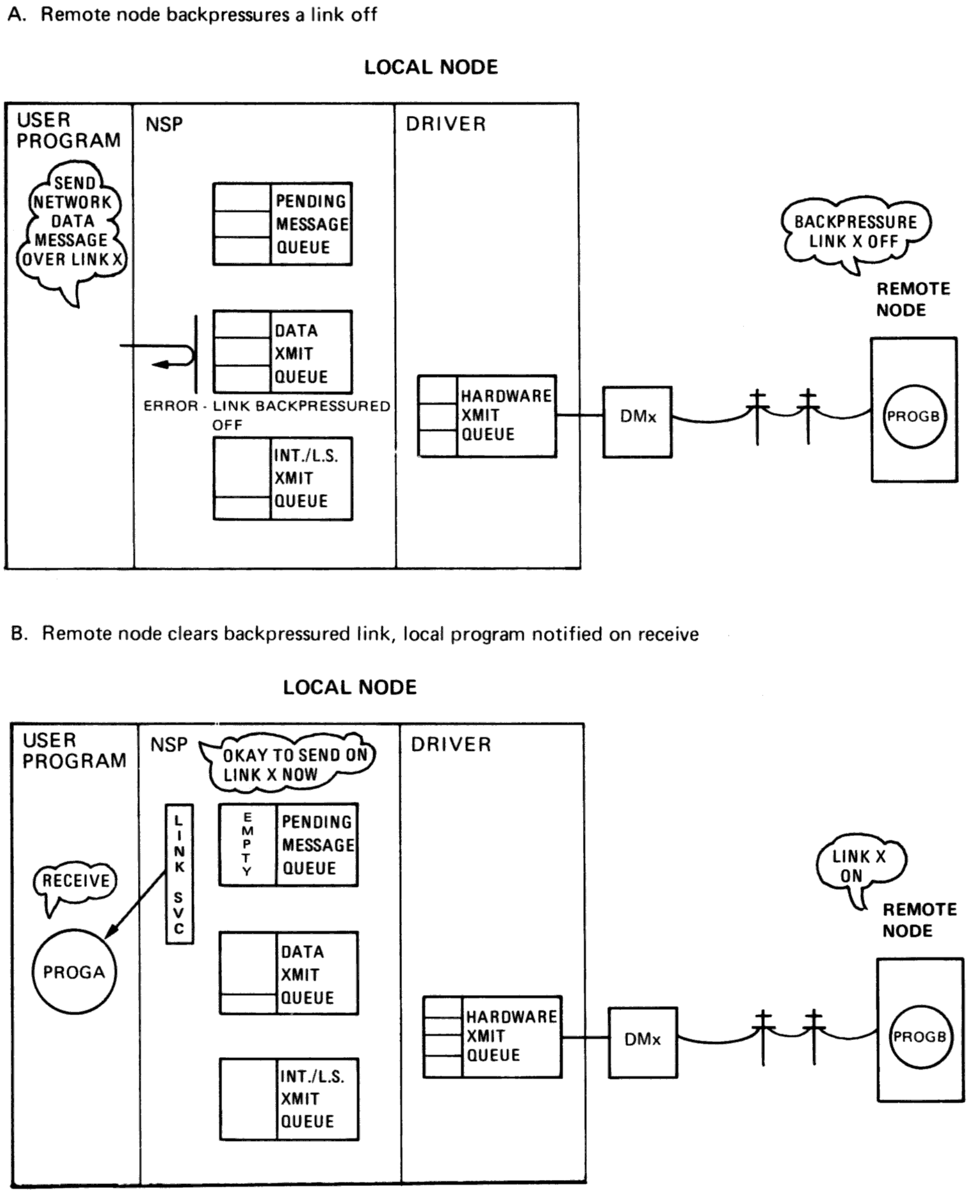


Figure 4–5: How Backpressure Flow Control Affects a DECnet/E Program as a Transmitter

4.2.3 Program-Regulated Flow Control

During the connection sequence that establishes a logical link, each program must select one of the following three flow control options to regulate incoming data over the link:

- **Segment Flow Control**. The program requests data from the remote program by indicating how many segments it will accept at any given time. A segment is defined as the amount of user data transferred in one Network Data Message. The maximum size of a segment is determined by the receive maximum established in the Connect Initiate or Connect Confirm Message. The sending program is not required to use this maximum, however, and can, in fact, send smaller segments. Note also that segments transmitted and received by the same program are not necessarily equal in size.
- **Logical Message Flow Control**. The program requests data from the remote program by indicating how many logical messages it will accept at any given time. A logical message can consist of one or more data segments. Control flags, included as part of a Network Data Message, indicate whether the data segment is the beginning, middle, end, or sole segment of a logical message.
- **No Active Flow Control**. The program sets no limits on the rate at which messages can be sent to it over a logical link.

If the program selects segment flow control or logical message flow control, the remote NSP will not send Network Data Messages over the link until the local program requests such transmission. Requests are made by sending a Link Service Message requesting a specific number of segments or logical messages.

If the program selects the third option, no active flow control, no Link Service Messages can be sent except those requesting Interrupt Messages. Network Data Messages will be transmitted by the remote NSP as the remote program sends them, subject to backpressure.

Flow control selection is independent at both ends of the logical link. That is, the remote and local programs need not select the same type of flow control. The option selected by the remote program, however, affects how the local program can send Network Data Messages. Therefore, the following discussion of flow control is presented from two viewpoints:

1. How the option selected by the local program affects incoming Network Data Messages
2. How the option selected by the remote program affects the way the local program sends Network Data Messages

Flow Control on Incoming Data

If the local program selects logical message flow control in the Connect Initiate or Connect Confirm Message establishing a logical link, the local program must request data from the remote program by sending Link Service Messages specifying the number of logical messages to be transmitted.

When the Send Link Service Message call is issued, the local NSP adds the value specified to a message counter maintained for the link. The local NSP also sends the information on to the remote NSP. The remote NSP likewise adds the value specified to a message counter it maintains for the link. Thus both NSPs count the outstanding message requests. Each time the remote NSP sends the end segment of a logical message, it decrements the message counter. When the counter reaches zero, the remote NSP will inhibit transmission until the local program sends another Link Service Message specifying another value to add to the message counter.

The local NSP uses its message counter as a protective measure. Each time it receives a data segment with the end-of-logical-message flag set, it decrements the message counter. When the counter reaches zero, the local NSP no longer accepts data segments over the link. Should the remote NSP send a segment while the local NSP's counter is zero, the message is discarded and the link is aborted. The local NSP places a Link Abort Message in the local program's queue of pending messages and sends a Link Abort Message to the remote NSP. (This situation is a protocol violation and should not occur with correct NSP operation.)

When the local program selects segment flow control in the Connect Initiate or Connect Confirm Message, the local and remote NSP maintain segment counters for the logical link. The remote program can still send segments with the logical message control flags set for the local program to process internally, but the counters maintained by NSP are kept for data segments sent, not for logical messages. Data is requested by sending a Link Service Message indicating the maximum number of segments to be sent. The local and remote NSPs add this value to their data segment counters and then decrement the counters as the remote program sends segments. When the data segment counter reaches zero, transmission over the logical link is suspended until another Link Service Message increments the count.

When no active flow control is requested in the Connect Initiate or Connect Confirm Message, segments from the remote program can be transmitted over the logical link without being requested by the local program. The local and remote NSPs do not keep counters to regulate data sent and Link Service calls are not used to request transmission of data. Data Message flow can still be restricted by backpressure, however.

Flow Control on Outgoing Data

When the remote program specifies a flow control option in its Connect Initiate or Connect Confirm Message establishing the logical link, the local NSP notes the option and establishes the appropriate counter. The counter is increased when a request for data is received from the remote system or program. If the remote program requested segment flow control, the counter is decremented by one each time the local program issues a Send Network Data Message call. If the remote program requested logical message flow control, the message counter is decremented by one each time a Network Data Message is sent that is flagged as the end data segment of a logical message.

When the counter equals zero, the local NSP inhibits further transmission of Network Data Messages over the logical link, returning an error if the local

program tries to do so. In this circumstance, the local NSP delivers a Link Service Message to the local program when the condition clears and the pending message queue is empty. A Link Service Message is not delivered, however, if the local program did not attempt to send a Network Data Message while the request count was zero.

This process is illustrated in Figure 4-6. The remote program has requested segment flow control. First, the remote program requests two segments (see A, in Figure 4-6). The segment counter for the remote program is incremented, but since PROGA has not had transmission inhibited, no Link Service Message is queued. The local program issues three Send Network Data Message calls (see B, in Figure 4-6). The segment counter is decremented to one with the first send, and to zero with the second send. The third send, while the counter is zero, results in an error. When the remote program receives and processes its two segments, it asks for two more segments (see C, in Figure 4-6). The segment counter is incremented from 0 to 2, reenabling data sends. Since PROGA attempted a send that failed, a Link Service Message is delivered to PROGA to indicate that data flow over the link has been reenabled. The Link Service Message is delivered to PROGA when it executes a Receive call when the pending message queue is empty.

4.2.4 Interrupt Message Flow Control

The transmission of Interrupt Messages over a logical link is subject to flow control by a request count mechanism. This mechanism is controlled by either the remote NSP or the remote program, depending on the particular DECnet implementation.

All logical links are initiated with an outstanding Interrupt Message request count of one from the remote system. If the local program sends an Interrupt Message over the logical link, this count is decremented to zero. The local NSP does not allow another Interrupt Message to be sent over this logical link until the remote NSP or remote program sends a Link Service Message, incrementing the request count. An attempt by the local program to send an Interrupt Message while the counter is zero will result in an error.

Once the request count is incremented by the remote system or program, the local NSP reenables Interrupt Message flow. If the local program has tried to send an Interrupt Message while the count was zero, NSP notifies the local program with a Link Service Message that Interrupt Messages can again be sent over this logical link. This Link Service Message is delivered to the local program on a Receive call when the pending message queue is empty. If the local program has not tried to send an Interrupt Message over the link while flow was disabled, no Link Service Message is delivered.

From the reverse viewpoint, all DECnet/E logical links are initiated with an outstanding request for one Interrupt Message. If the remote program sends an Interrupt Message over the link, the local NSP decrements this count to zero. No further Interrupt Messages are accepted until the local program requests one by sending a Link Service Message to increment the count. This also increments the request counter at the remote end and thus allows the

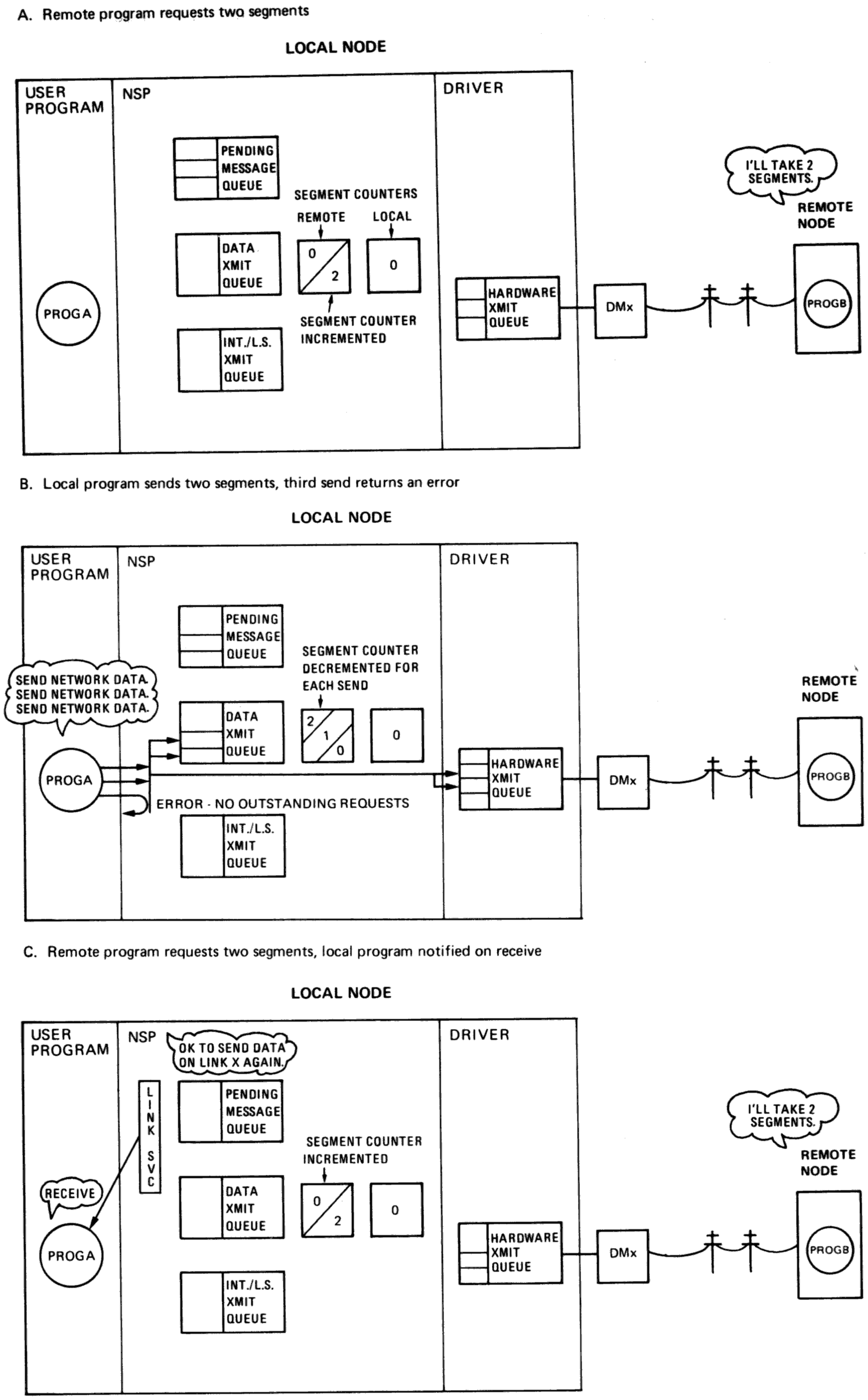


Figure 4–6: How the Flow Control Option Selected by the Remote Program Affects a DECnet/E Program as a Transmitter

remote program to send an Interrupt Message if it so desires. DECnet/E NSP does not allow the local program to request an Interrupt Message for a logical link unless a previous Interrupt Message on the link has been received.

In DECnet/E, only one Interrupt Message can be requested on a logical link at any one time. That is, the request count is either zero or one. If a program tries to issue a Link Service Message requesting an Interrupt Message on a link when the request count is already one, the Link Service Message will result in an error.

Figure 4–7 illustrates Interrupt Message flow control for a DECnet/E receiver. The remote program sends an Interrupt Message over a logical link (see A, in Figure 4–7). The local NSP decrements the Interrupt Message counter to zero and does not accept further Interrupt Messages over this link. The remote NSP also decrements its own counter and inhibits further Interrupt Messages from being sent. The local NSP maintains its counter as a safeguard measure. The local program sends a Link Service Message requesting one Interrupt Message over this logical link (see B, in Figure 4–7). The local NSP increments its counter and tells the remote NSP to reenable Interrupt Messages over the link. The local program must receive the first Interrupt Message before the Link Service Message can be sent.

4.2.5 Programming Hints for Flow Control

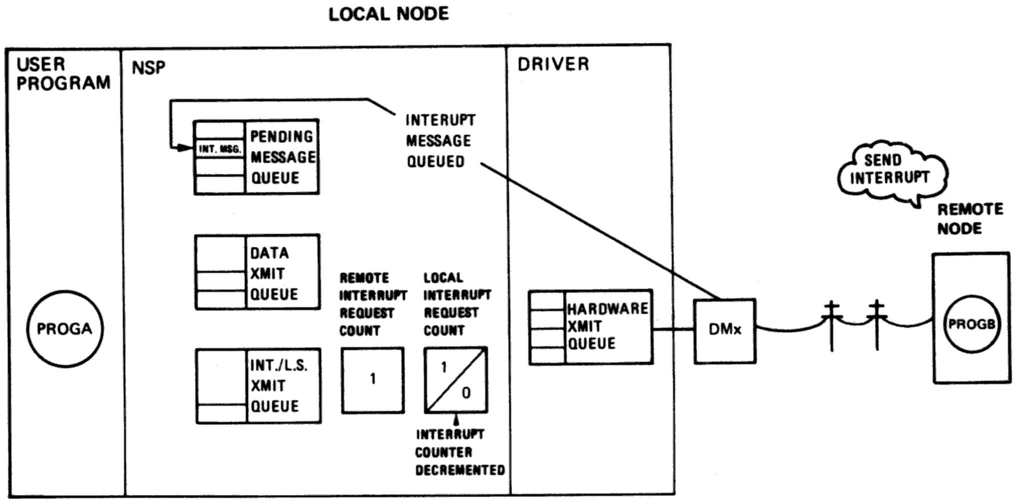
A DECnet/E program can be prevented from transmitting for a number of reasons:

1. No buffer space is available at the local node. (Prohibits sending of all message types.)
2. The Data transmit queue is full. (Prohibits sending of Data Messages.)
3. The Interrupt/Link Service transmit queue is full. (Prohibits sending of Interrupt and Link Service Messages.)
4. Message flow over the link has been inhibited by the remote system because of backpressure condition. (Prohibits sending of Data Messages.)
5. No requests are outstanding for segments or logical messages from the remote program. (Prohibits sending of Data Messages.)
6. No requests are outstanding for Interrupt Messages. (Prohibits sending of Interrupt Messages.)

The first three conditions indicate congestion at the local node. They are temporary and no notification is given to the program when the condition clears. When one of these errors occurs on a send call, the program should simply reissue the call after a short delay of one to five seconds.

The last three conditions, however, indicate congestion at the remote node. Since there is no way to predict how long it can take for one of these conditions to clear, the local NSP does not allow further send requests of Data or Interrupt Messages until the remote system indicates that the condition has cleared.

A. Remote program sends an Interrupt Message over a logical link



B. Local program reenables interrupts for the link (allowed only when previous Interrupt Message for this logical link has been received)

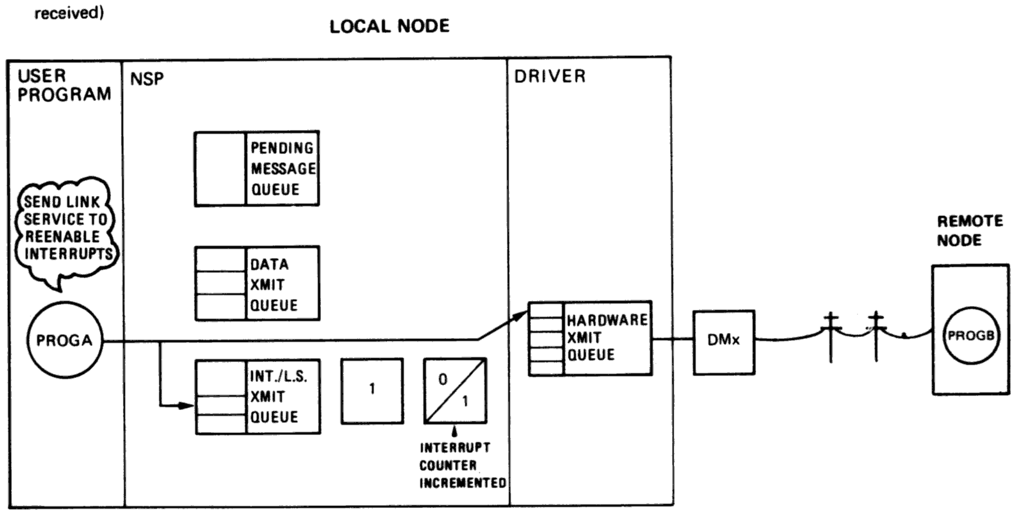


Figure 4–7: How Interrupt Message Flow Control Affects a DECnet/E Program as a Receiver

When sending is blocked for one of these last three conditions, any attempt to send a message by the local program results in an error. The program can still receive messages, however. If an attempt to send a message resulted in an error in this fashion, NSP notifies the program when the condition clears. It does this by delivering a Link Service Message to the program when the program issues a Receive call and the pending message queue is empty. No notification is given if the local program did not try to send a message.

Thus a program whose main function is to transmit data can be designed to send data over a logical link until transmission is blocked by one of the last three conditions. When sending is inhibited, the program can issue Receive calls to process all pending messages. When the pending message queue is empty and all conditions that block flow have cleared, a Link Service Message

is delivered on a Receive call, giving current status information on flow control for the logical link. (If more than one logical link has been blocked, Receive calls should be issued for all links until no messages are returned, since link service information can also be available on the other links.)

A program whose main function is to receive data is concerned with keeping the pending message queue emptied so that the queue does not fill and cause the local NSP to inhibit incoming message flow over the program's logical links. The Receive call with sleep can be used effectively by such a program so that it will be notified when a message is delivered to the queue. In this case, a second Receive call must be issued to retrieve the message.

An interactive program can choose between these mostly-send and mostly-receive approaches, according to the requirements of the application. A program can also take advantage of the link status information that can be returned after sending Network Data, Interrupt, and Link Service Messages to regulate send and receive requests. The link status information returned is in the same format as a received Link Service Message.

Part III
Network Programming in BASIC–PLUS

Chapter 5
Network Programming in BASIC–PLUS

This chapter presents the specific system calls, with detailed formats, used to access the DECnet/E message services in the BASIC–PLUS programming language.

5.1 Programming Background

In BASIC–PLUS, the system calls for message services, both local and DECnet, are coded as SYS calls to the RSTS/E File Processor (FIP).

5.1.1 General Format of SYS Calls

The general format of the message SYS calls in BASIC–PLUS is:

v$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(*s*%) + ...)

where

v$	is the name of a string that will receive control information returned by some of the calls — the *target* string. (The dollar sign indicates a character string variable.)
=	is the assignment operator.
SYS(...)	indicates a system call.
+	is the concatenation operator required between function, subfunction, and argument values.
CHR$(6%)	is the system function code for a call to the File Processor (FIP). (The CHR$ function converts the integer value 6 — the % indicates an integer in BASIC–PLUS — to the character string format required in SYS calls.)

CHR$(22%) is the send/receive function code.

CHR$($s$%) is the user-specified subfunction code. (For example, $s = 1$ indicates a Declare Receiver system call; $s = -2$ indicates a Send Connect Initiate Message system call.)

... indicates other arguments required for the various system calls. These arguments must be specified in character string format. The CHR$($x$%) construction can be used for one-byte integer values. CHR$($x$%) + CHR$(SWAP%(x%)) can be used for two-byte integers. ASCII values are enclosed in quotation marks. (Useful coding conventions for SYS calls are described in Chapter 7 of the *RSTS/E Programming Manual.*)

5.1.2 Buffer Allocation for Message Operations

Local and DECnet communication normally requires the use of buffer areas to hold user data being sent to another program or to receive data from another program. The send calls allow the data to be sent from a buffer or appended to the SYS call string. Buffers are usually required to receive data.

Opening any device on any I/O channel allocates a buffer for I/O to or from the device. There are twelve I/O channels available to each BASIC-PLUS program running under RSTS/E, referenced by the numbers 1 through 12. A convenient way to allocate a buffer for message operations is to open the null device on any channel, using the RECORDSIZE option to specify the length of the buffer desired. For example,

```
OPEN "NL:" AS FILE 12%, RECORDSIZE 128%
```

would allocate one 128-byte buffer. The null device is used because it is always available and can be opened as many times as necessary (up to 12) to obtain required buffers. Also, opening the null device does not imply any side effects as sometimes occurs when opening a real device.

In the send/receive SYS calls, the buffer is referenced by the channel number — CHR$(12%) in the previous example. Channel 0 can also be used if 128 is added to the channel number: for example, CHR$(128%+0%). (This, in fact, works for all channel numbers: CHR$(128%+x%), for x = 0 to 12.) Data in the buffer can be referenced with FIELD statements or moved with standard LSET and RSET statements in BASIC-PLUS.

5.1.3 Notation Used in This Chapter

The SYS call descriptions in this chapter are organized as separate subsections. The heading gives the name of the call and the word "Privileged" if the call can be issued only from a privileged program. ("Privilege" is a special condition for a user job under RSTS/E, as defined in the *RSTS/E Programming Manual.*) "Local" or "DECnet" (or both) on the heading line indicates whether the call is used for local or network message services.

The first page of each call description consists of an introductory paragraph, followed by a table showing the specific format of the data passed and returned for each call. The first column of the table indicates the byte position of each field within the string. The second column indicates the value of the field, and the third column gives a brief description of the way the value is used. Lowercase items in italics are to be supplied by the user. The following forms appear:

CHR$(6%)	indicates a one-byte field, in character string format, that must have the value shown (6 in this particular case).
CHR$(*x*%)	indicates a one-byte field, in character string format, with *x* being an integer constant or variable name supplied by the user.
CHR$(0%) or 0%	indicates that the integer value zero appears, in character string format, in all bytes of the field. Zeros must appear in reserved fields. It is also good practice to fill ignored fields with zeros. This could be coded as CHR$(0%) for one-byte fields or as STRINGS$(n%,0%) for n-byte fields.
x%	indicates a two-byte field, with *x* being an integer constant or variable supplied by the user. Such variables can be converted to character string format using the construction CHR$(*x*%) + CHR$(SWAP%(*x*%)).
abc$	indicates a character string value — either a constant enclosed in quotes, such as "STRING", or a variable name that has been previously defined to contain the desired string value.

The format table is followed by an expanded discussion of the user-supplied arguments. A description of possible errors that can occur for each call is given next, followed by a brief example of each call.

5.2 Declare Receiver — Privileged (Local and DECnet)

The Declare Receiver call must be executed before a program can receive any messages or send any network messages. (Local messages can be sent without first executing this call.) The call defines an identifying object type or name (or both) and any restrictions on incoming messages. The monitor associates this information with the RSTS/E job number, setting up a Receiver ID Block for the job as described in Section 3.2.3. Only one Declare Receiver call can be in effect at any one time for a particular job. It remains in effect until a Remove Receiver call is issued for the job.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(1%)	Declare Receiver subfunction code.
4	CHR$(0%)	Reserved - must be zero.
5–10	*name$*	Logical name: 1 to 6 characters. Left-justified with space fill to 6 characters. Or null if network access by object type only.
11–12	0%	Reserved - must be zero.
13–16	0%	Ignored.
17–20	0%	Reserved - must be zero.
21	CHR$(*objtyp%*)	Object type code, 0 to 255: If $\neq 0$, this value can be used by remote programs to address the calling program.
22	CHR$(*access%*)	Access control field = $l\% + p\% + n\% + o\% + s\%$: *l* Local senders? 0 = no, 1 = yes *p* Privileged local senders? 0 = no, 2 = yes *n* Incoming logical links? 0 = no, 4 = yes *o* 1 logical link/execution? 8 = no, 0 = yes *s* Sleep on pending message? 0 = no, 16 = yes
23–24	*bmax%*	Buffer maximum, 1 to 32767 bytes: Maximum amount of monitor's buffer pool space to be used for pending local message data. Negative or 0 value indicates no such space is to be used for local message data.
25	CHR$(*mmax%*)	Message maximum, 0 to 255: Incoming local messages are not queued if the total number of pending messages for the program is greater than this value.
26	CHR$(*lmax%*)	Link maximum, 0 to 63: Incoming requests for logical links are rejected if the total number of links for the program is greater than this value.
27–28	0%	Ignored.
29–40	0%	Reserved - must be zero if passed.

Data Returned:

No meaningful data is returned.

Discussion:

name

The logical name is a string containing one to six characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underline). The name is left-justified in the string and (if necessary) padded to six characters with spaces. Leading or embedded spaces are invalid. If no name is to be specified (that is, the logical name is null), the string should consist of 6 spaces.

The name is used to identify the calling program for queuing messages from local or network programs. Local programs can address the calling program by the logical name. Network programs can use either the logical name or the object type code, if one is declared.

If the logical name is null, only network access by object type is allowed (see discussion of *name*, *objtyp*, and *access*). Otherwise, the logical name must be nonnull and unique.

objtyp

An object type code is another form of network addressing. As discussed in Section 3.2.1, the object type defines some service that the program performs. If the calling program is addressed by object code alone (*name* is null and *access* indicates network access only), multiple copies of the program can be executed simultaneously. Unlike *name*, the object type code need not be unique — multiple copies of a program can declare their identity with the same type code simultaneously. If *name* is null, *objtyp* cannot be zero. (Network addressing is discussed in detail in Section 4.1.)

Acceptable values for the object type code range from 0 through 255. The value 0 indicates that object code addressing is not being used for the calling program. The range from 1 to 127 is reserved for DECnet use (see Appendix A). The range from 128 to 255 is available for definition and use by a network installation.

access

This field contains the sum of five bit values ($l\% + p\% + n\% + s\% + o\%$) that are used to determine access to the declaring program and to modify certain aspects of the message operations.

Three of these bit values (l, p, and n) limit the types of senders that are allowed to queue messages for the calling program. They do not, however, limit the kind of messages that can be sent.

If $l = 0$, messages from local senders will not be queued for the calling program. (Local senders who use the network functions are considered network senders in this context.) If $l = 1$, messages from local senders will be queued.

If $p = 0$, incoming messages from both local privileged and local nonprivileged senders will be queued for the calling program. If $p = 2$, incoming messages from local senders will be queued only if the sender is privileged. (This bit is ignored if $l = 0$, that is, if no local senders are allowed.)

If $n = 0$, incoming requests for logical links are rejected by NSP. If $n = 4$, incoming Connect Initiate Messages are passed on to the receiver, subject to the declared link maximum (*lmax*). In this case, the program must provide its own protection from unauthorized access, if necessary.

Table 5–1 summarizes access code values and the access permitted for each.

Table 5–1: Types of Receiver Access

Access Code (n%+p%+l%)	Network Logical Links	Local Senders Privileged	Local Senders Allowed
0	no	no	no
1	no	no	yes
2	no	no	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	no	no
7	yes	yes	yes

Bit value o is used to regulate single-link programs, modifying the function of the program's declared link maximum (*lmax*). If $o = 0$, incoming requests for logical links do not affect the program's link maximum. If $o = 8$, however, after queuing an incoming Connect Initiate Message to the receiver, NSP sets the program's link maximum to zero, inhibiting all further incoming connect requests. This, in effect, modifies the meaning of the link maximum from "one link at a time" to "one link per program execution."

By setting this "one-shot" bit to 8, a program prevents the possibility that NSP will queue a Connect Initiate Message for it after its one logical link is disconnected and before it issues a Remove Receiver call. For example, the DECnet/E FAL utility sets the one-shot bit to 8 and declares a link maximum of 1. Once FAL is started as the result of an incoming request from a remote NFT utility, NSP queues the Connect Initiate Message and zeros the link maximum. Thus, there is no possibility that another incoming connect request could be queued for that copy between the time that NFT breaks the first link and the time that FAL issues its Remove Receiver call. This ensures that each incoming connect request starts a new copy of FAL.

Bit value s is used to modify the function of the RSTS/E conditional Sleep monitor call. If $s = 0$, any unreceived messages queued for the program will block the execution of a conditional Sleep call. This is the normal RSTS/E

function. If $s = 16$, however, the monitor will not check the program's message queue when determining whether or not it should in fact suspend program execution. Several other conditions (such as a delimiter received on an open terminal) can still block the sleep, but a pending message will not.

name, objtyp, and *access*

The system checks the values of the *name*, *objtyp*, and *access* fields to ensure that they are consistent. For example, a null logical name and an access code allowing local senders would be inconsistent since local senders can address the program by logical name. The system validates these arguments as shown in Table 5–2. If the arguments are invalid, the system will reject the Declare Receiver call, returning an error (ERR = 18). If the arguments are valid, the system allocates a small buffer to hold the information passed. If no buffers are available, the call will fail with an error (ERR = 4). A retry may be successful.

Table 5–2: System Validation of Name, Object, and Access Parameters

Access Code (n%+p%+l%)	Senders Permitted	Object Type	Logical Name
0,2	None	Ignored	Ignored.
1,3	Local only	Ignored	Must be nonnull and unique.
4,6	Network only	Zero	Must be nonnull and unique.
		Nonzero	If the logical name is null (at least one leading binary zero byte), the only access permitted is by object type. Multiple copies of the program can coexist. If the logical name is nonnull, it must be unique. Network senders can refer to the calling program by name or by object type. Only one copy of the program using this logical name can execute at any given time.
5,7	Network/Local	Any value	Must be nonnull and unique so that local senders can refer to the calling program by logical name. Network senders can use name or object type (if object type $\neq 0$). Only one copy of the program using this logical name can execute at any given time.

bmax

Until a Receive system call is executed, a pending message occupies system buffer space. One 16-word buffer from the monitor's buffer pool is used for user- or DECnet-defined parameters and other system-specific information.

Additional buffer space for the message data is usually allocated from the extended buffer pool. If an extended pool does not exist or no space is available there, the monitor's buffer pool is used.

Because the monitor pool is a critical system resource, the receiver must set a limit on the amount of space to be used on its behalf. The *bmax* parameter sets a limit (1 to 32767 bytes) on the total monitor pool space to be used for the user data portion of messages.

When this buffer maximum is exceeded, Local Data Messages are no longer queued for the program. NSP will return an error to any local program that tries to send a Local Data Message to the program at this point. Network messages will be queued regardless of the declared buffer maximum but any monitor pool space used will be counted against the maximum.

A zero or negative value indicates that the monitor's buffer pool is not to be used for the user data portion of pending Local Data Messages.

mmax

The message maximum (*mmax*) limits the number of messages that will be queued for the calling program. The value can range from 0 to 255.

NSP keeps a count of the total number of messages queued for each declared receiver. When one of these counters equals or exceeds the message maximum set by the receiver:

1. Incoming Local Data Messages are no longer queued for the receiving program. An error is returned to the local sender.
2. Incoming Connect Initiate Messages cause NSP to determine if another copy of the program can be started automatically (see Section 4.1.2). If not, NSP rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.
3. An incoming Network Data Message causes NSP to inhibit incoming data messages on that particular logical link and to negatively acknowledge (NAK) the message. This forces the remote system to retransmit when flow is reenabled. (See Section 4.2.2 for further details on backpressure flow control.)

lmax

The link maximum (*lmax*) limits the number of incoming requests for logical links that will be queued for the calling program. The value can range from 0 to 63.

If the number of links currently active is greater than or equal to *lmax* when a remote Connect Initiate Message is received for the program, the local NSP determines if another copy of the program can be started automatically (see Section 4.1.2). If not, NSP rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

By declaring a small link maximum, a program can avoid the overhead of responding to remote Connect Initiate Messages when it is known beforehand that the program can only handle a limited number of logical links.

A zero value for *lmax* has the same effect as setting $n = 0$ in the access control field. That is, incoming requests for logical links are rejected by NSP. The *lmax* parameter does not limit the number of logical links that can be initiated by the program.

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
3	ACCOUNT OR DEVICE IN USE The calling job already exists in the system's list of declared receivers. This error can indicate a logic error in the program or that a previous program running under the same job number failed to remove itself from the receiver list before terminating. In the latter case, a Remove Receiver call should be issued, followed by another Declare Receiver call. It is common practice to issue a Remove Receiver call immediately before the Declare Receiver call.
4	NO ROOM FOR USER ON DEVICE There were no small buffers available to hold the arguments passed with the Declare Receiver call. Since the system's use of small buffers is dynamic, a retry may succeed.
10	PROTECTION VIOLATION The calling program must be privileged at the time the Declare Receiver call is issued.
16	NAME OR ACCOUNT NOW EXISTS The logical *name* passed in bytes 5–10 is being used by another job.
18	ILLEGAL SYS() USAGE An inconsistency was found in the arguments passed. See *name*, *object*, and *access* discussion.

Example:

The following example shows a receiver declaration allowing only network access via logical links. The logical name is ADMIN and no object type code is used. Since local messages are not an issue, the buffer maximum (BMAX) is set to zero. The message maximum (MMAX) is set to 10. If more than 10 messages accumulate in the program's pending message queue, an incoming Connect Initiate Message will cause NSP to send back a Connect Reject Message to the remote program and any further incoming Data Messages will cause NSP to inhibit incoming message flow on all the program's logical links because of the backpressure condition. Likewise, the link maximum (LMAX) is set to 1. When the program has established one logical link, an incoming Connect Initiate Message will cause NSP to send back a Connect Reject Message to the remote program.

```
LNAME$   = "ADMIN "
ACCESS% = 4%
BMAX%    = 0%
MMAX%    = 10%
LMAX%    = 1%
!
! DECLARE RECEIVER EXAMPLE
!
X$ = SYS(CHR$(6%)+CHR$(22%)+CHR$(1%)+CHR$(0%)
        +LNAME$ + STRING$(10%,0%)
        +CHR$(0%)
        +CHR$(ACCESS%)
        +CHR$(BMAX%) + CHR$(SWAP%(BMAX%))
        +CHR$(MMAX%)
        +CHR$(LMAX%))
```

5.3 Remove Receiver (Local and DECnet)

The Remove Receiver system call removes the specified job from the system's list of declared receivers. All pending messages are discarded and any active logical links are aborted. This call should be executed at the completion of message processing. This prevents unwanted messages from accumulating in the queue of pending messages.

Privileged programs can remove other RSTS/E jobs with this call, although normally only system utility programs would do this.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(0%)	Remove Receiver subfunction code.
4	CHR$(0%) or CHR$(*jobx2%*)	Zero, to remove the calling job (normal usage), or Job number times two, to remove another local RSTS/E job. Only privileged programs can issue this call with byte 4 $\neq 0$.
5–12	0%	Reserved - must be zero if passed.
13–16	0%	Ignored.
17–40	0%	Reserved - must be zero if passed.

Data Returned:

No meaningful data is returned.

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
10	PROTECTION VIOLATION The caller is nonprivileged and has attempted to remove another job (that is, byte 4 is nonzero).
18	ILLEGAL SYS() USAGE The number passed in byte 4 was odd. Byte 4 must be zero to remove the calling job, or job number times two to remove another job.

Example:

The following example shows a typical Remove Receiver call:

```
!
! EXAMPLE OF REMOVE RECEIVER
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(0%))
```

5.4 Get Local Node Parameters (DECnet)

The Get Local Node Parameters system call returns information to the calling program concerning the local node's network parameters. Data returned includes the node's name, address and alias (if any), and the default user's account to be used when an incoming Connect Initiate Message does not specify accounting information. (See the *DECnet/E System Manager's Guide* for a discussion of alias and Section 4.1.4 of this manual for a discussion of the default user's account.)

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-19%)	Get Local Node Parameters subfunction code.
4	CHR$(1%)	Must be set to "1".
5-40	0%	Reserved - must be zero.

Data Returned:

The local node network parameters are returned to the calling program in the target string. The format of the information is as follows:

Bytes	Value	Description
1-2	—	Not meaningful - should be ignored.
3	CHR$(-19%)	Get Local Node Parameters subfunction code.
4	—	Not meaningful - should be ignored.
5-10	*name$*	Local node name: 1 to 6 characters. Left-justified with space fill to 6 characters.
11-20	—	Not meaningful - should be ignored.
21-22	*number%*	Local node number; 1 to 255.
23-28	*alias$*	Local node alias: 1 to 6 characters. Left-justified with space fill to 6 characters. If no alias is defined, 6 spaces are returned.
29-30	*ppn%*	Default project-programmer number: Byte 29 = programmer number, byte 30 = project number. Or both zero if none defined.
31-40	—	Not meaningful - should be ignored.

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until NSP is enabled by the system manager.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example illustrates the Get Local Node Parameters call:

```
!
! EXAMPLE OF GET LOCAL NODE PARAMETERS
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-19%) + CHR$(1%))
```

5.5 Log User Event — Privileged (DECnet)

The Log User Event call permits a user-written program to queue an event to the system event processor for logging. Before the event is logged, it is time-stamped by the system in the standard Network Information and Control Exchange (NICE) protocol format. The system manager can use the normal network management SET and CLEAR LOGGING commands to control the filtering of these events. Optional data can accompany a user event.

NOTE

This system call performs a highly specialized function which requires a great deal of special knowledge of DECnet and the DIGITAL Network Architecture (DNA). It is provided as a convenience for the sophisticated network user and is not intended for normal network programs.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-10%)	Log User Event subfunction code.
4	CHR$(*evmod%*)	Error return modifier. If 1, log "missed event" message if event queue is full; if 0, return NOROOM status return.
5–6	*evcode%*	Event code representing the class (*evcls%*) and type (*evtyp%*) of the user event. This two-byte integer is formed as follows: *evcode* = *evcls* * 64% + *evtyp*
7–10	0%	Reserved - must be zero.
11	CHR$(*chnl%*)	Channel number of buffer containing the optional user data to be passed to the event logger for processing. If 0, this data is passed as part of the string.
12	CHR$(0%)	Reserved - must be zero.
13–14	*length%*	Length, in bytes, of data to be passed from the I/O buffer, 0 to 200. If 0, the number of bytes equals the buffer size minus the *offset*. Ignored if *chnl* = 0.
15–16	*offset%*	Offset, in bytes, from the beginning of the buffer. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17–20	0%	Reserved - must be zero.
21–22	*enval%*	Identifier of the entity with which this event is concerned.
23	CHR$(*entyp%*)	Type of entity with which this event is concerned.
24–40	0%	Reserved - must be zero.
41+	*data%*	Optional user data, if sent as part of the string. Maximum of 200 bytes. Ignored if *chnl* ≠ 0.

Data Returned:

No meaningful data is returned.

Discussion:

evmod

This parameter specifies what the system should do if it is unable to log the user event due to lack of resources. If *evmod* is set to zero, the call terminates with an error of NOROOM (ERR = 4); the program can retry logging the event after a short delay. If *evmod* = 1, the system logs a "missed event" event message, and the call completes successfully.

Note that in the second case, the user data is not logged. The system merely logs the fact that it has lost some unidentified data. This is identical to the procedure followed with DECnet internal events when the event queue is full or there are no more small buffers.

evcode

This parameter is a two-byte integer representing the class (*evcls%*) and type (*evtyp%*) of the user event to be logged. The class and type are packed into one two-byte value as follows:

evcode = *evcls* * 64% + *evtyp*

Events are divided into classes, according to the DNA layer from which they originate. Classes 480 to 511 are reserved for customer-specific events. However, DECnet/E supports only class 480. Thus, *evcls* must be 480. The event type, *evtyp*, must be in the range of 0 to 31.

enval

This parameter associates a value to the entity specified by the *entyp* parameter. If the entity is a node, *enval* should be set to the node address. If the entity is a line or a circuit, *enval* should be set to the address of the Device Data Block (DDB) maintained for the device.

entyp

This parameter specifies the entity type with which the event is concerned. There are four entity types supported by DECnet/E, defined as follows:

-1	No specific entity
0	Node
1	Line
3	Circuit

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which channel buffer contains the user data to be processed with the event. Or it can be 0, indicating that this information is appended to the SYS call string.

If this data is included, it must be in NICE protocol format. Basically, this means that the data should be formatted in a 3-part field: a parameter type, a data type, and a parameter value. DECnet parameter types can be used if the event concerns a node, a circuit, or a line. In this case, the event logger will output standard descriptive text for the specific parameter. If a DECnet parameter type is not used, parameter type in the range of 3900 to 4095 can be specified. (See the *DIGITAL Network Architecture, Network Management Functional Specification*, Version 3.0.0, for further information on event definitions and parameters.)

Up to 200 bytes of data can be passed to the event logger for processing.

length

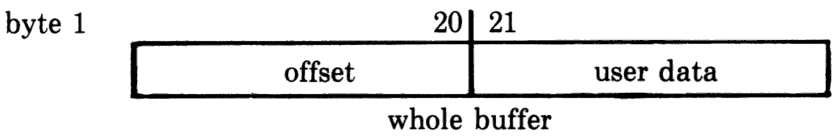
The *length* parameter indicates how many bytes are to be processed from the channel buffer. Up to 200 bytes of data can be passed.

If *length* = 0, the number of bytes processed is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes passed to the event processor is equal to the buffer size minus *offset*. This size must also be less than or equal to 200.

The *length* parameter is ignored if *chnl* = 0; that is, if the data is appended to the SYS call string.

offset

If an I/O buffer is used, the user data is assumed to be in the buffer in contiguous bytes. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

data

The event data to be passed to the event logger for processing can be included as part of the SYS call string, rather than in an I/O buffer. If this is the case, the data is passed in these bytes.

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.

(continued on next page)

ERR Value	Error Text and Meaning
4	NO ROOM FOR USER ON DEVICE The event queue is full or there are no small buffers available to hold the event. A retry may succeed. (This error is returned only if *evmod* = 0.)
5	CAN'T FIND FILE OR ACCOUNT The event logger is not running. Have the system manager start the event logger.
10	PROTECTION VIOLATION The calling program must be privileged at the time the Log User Event system call is issued.
18	ILLEGAL SYS() USAGE The parameters passed are either inconsistent or invalid.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example illustrates the logging of a user event, class 480. and type 3. The entity node number is 135. The optional data passed for processing is part of the SYS call string and consists of an *lla* parameter number (2130) and value. A "missed event" is logged if the event itself cannot be.

```
EVMOD%   = 1%
EVCLS%   = 480%
EVTYP%   = 3%
EVCODE%  = EVCLS% * 64% + EVTYP%
ENTYP%   = 0%
ENVAL%   = 135%
PRMTYP%  = 213%
DATTYP%  = 2%
DATVAL%  = 40960%
DATA%    = CHR$(PRMTYP%) + CHR$(SWAP%(PRMTYP%)) + CHR$(DAPTYP%)
         + CHR$(DATVAL%) + CHR$(SWAP%(DATVAL%))
!
! EXAMPLE OF LOG USER EVENT CALL
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-10%)
       + CHR$(EVMOD%)
       + CHR$(EVCODE%) + CHR$(SWAP%(EVCODE%))
       + STRING$(14%,0%)
       + CHR$(ENVAL%) + CHR$(SWAP%(ENVAL%))
       + CHR$(ENTYP%)
       + STRING$(17%,0%)
       + DATA$
```

The Log User Event system call in the previous example generates the following event message:

```
Event type 480.3
Occurred 14-Dec-81 15:31:39.7 on node 135 (GROK)
Node address 135 (GROK)
Parameter #2130 = 40960
```

5.6 Send System Calls

There are nine separate send calls — one local and eight network. Each of these calls causes NSP to format a message that is then either queued to a local program or transmitted across the network to a remote program.

5.6.1 Send Local Data Message (Local)

The Send Local Data Message system call sends up to 532 bytes of user data to another program in the local RSTS/E environment. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, the calling program must be privileged. The receiving program is identified in the call by either its job number or by the logical name with which it declared itself a receiver.

Up to 512 bytes of user data can be sent from a channel buffer or appended to the string. An additional 20 bytes of user data can be sent from the string.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-1%)	Send Local Data Message subfunction code.
4	CHR$(*jobx2%*)	Job number (times two) of the local job to receive the message. If 0, *name* (bytes 5–10) is used to identify the intended receiver.
5–10	*name$*	Declared logical name of local program to receive the message. 1 to 6 characters, left-justified, with space fill. Used if *jobx2* = 0, ignored otherwise.
11	CHR$(*chnl%*)	Channel number of buffer containing the message data, 1 to 12. If 0, the data is to be sent as part of the string.
12	CHR$(0%)	Reserved – must be zero.
13–14	*length%*	Length, in bytes, of the message data to be sent from the buffer, 1 to 512. If 0, the number of bytes sent equals buffer size minus *offset*. Ignored if *chnl* = 0.
15–16	*offset%*	Offset, in bytes, from the beginning of the buffer where the message data begins. If = 0, the message data is assumed to start at beginning of the buffer. Ignored if *chnl* = 0.
17–20	0%	Reserved – must be zero.
21–40	*param$*	Optional user parameter string. Up to 20 bytes of user data can be passed here as part of the string.
41+	*data$*	0 to 512 bytes of message data to be sent to the local program, if sent as part of the string (that is, if *chnl* = 0). Ignored if *chnl* ≠ 0.

Data Returned:

No meaningful data is returned.

Discussion:

jobx2

If $jobx2 = 0$, the message data is sent to the program identified by the logical name field. Otherwise, the message data is sent to the program whose job number (times two) is equal to $jobx2$. For example, if $jobx2 = 8$, the message is sent to job 4.

name

The logical name identifies the intended local receiver when the *jobx2* parameter = 0. The intended receiver must have declared its identity with this name in its Declare Receiver call. The name consists of one to six characters, left-justified in the field and, if necessary, padded to six characters with spaces. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underline). Leading or embedded spaces are invalid. This field is ignored when $jobx2 \neq 0$.

chnl

The *chnl* parameter is either an integer from 1 to 12, indicating which I/O buffer contains the message data, or the value zero, indicating that the message data is part of the SYS call string.

length

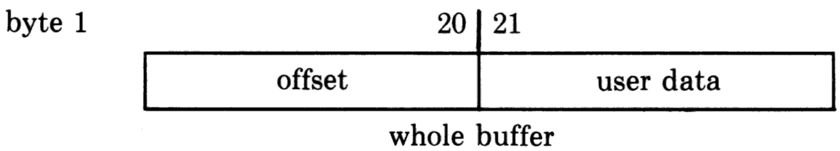
The *length* parameter indicates how many bytes are to be sent from the I/O buffer. The maximum value for *length* is 512. If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement that allocated the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This size must also be in the range of 1 to 512.

The length field is ignored if *chnl* = 0, that is, if the user message data is appended to the SYS call string.

offset

If an I/O buffer is used, the user data can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with the *offset* parameter. For example, if *offset* = 20, the user data begins in byte 21:

The *offset* field is ignored if *chnl* = 0.



param

The optional *param* string allows up to 20 additional bytes of user data to be passed. It is delivered to the local program separate from the message data. For example, when a BASIC–PLUS program issues a Receive call that returns

a Local Data Message, the parameter data is returned in the target string for the Receive call and the message data is returned in the channel buffer specified in the Receive call.

data

The user data (if any) can be sent from the SYS call string (*data*) or from the I/O buffer specified by the *chnl* parameter. Up to 512 bytes can be sent in either case. If a buffer is used, remember that *length* bytes will be transmitted and that the first *offset* bytes in the buffer will not be sent.

Possible Errors:

ERR Value	Error Text and Meaning
4	NO ROOM FOR USER ON DEVICE The number of pending messages for the intended local receiver is at its declared maximum. The calling program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	CAN'T FIND FILE OR ACCOUNT The intended local receiver could not be located in the system's list of declared receivers. The receiver must be declared (with a Declare Receiver call) before any data can be transmitted to it.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 of the data passed is not open. The program must OPEN the channel to allocate a buffer for message operations.
10	PROTECTION VIOLATION Some access violation has occurred. Either the receiver does not allow any local senders or the sender is nonprivileged and the receiver allows only privileged senders.
18	ILLEGAL SYS() USAGE The quantity in byte 4 (*jobx2*) is odd. Byte 4 must be 0 or the receiver's job number times two.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13-16 are inconsistent. The following must be true: 1. The *offset* must be less than the buffer size. 2. The *length* must be less than or equal to the buffer size minus *offset*. 3. The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be less than or equal to 512. The *offset* and *length* fields are checked for validity whenever *chnl* (byte 11) $\neq 0$.
32	NO BUFFER SPACE AVAILABLE System buffers are currently not available to store the message for the intended local receiver. A later retry may proceed without error.

Example:

This example shows a local send to a program named "HARVEY". The user data is sent from the SYS call string, beginning in byte 41.

```
MSG$ = "THIS MESSAGE IS SENT FROM THE SYS CALL STRING"
!
! EXAMPLE OF LOCAL DATA SEND
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-1%) + CHR$(0%)
        + "HARVEY" + STRING$(30%,0%)
        + MSG$)
```

5.6.2 Send Connect Initiate Message (DECnet)

The Send Connect Initiate Message call requests a logical link to another program. The other program is identified with a Connect Data Block and the user link address by which the calling program will refer to the link is established. The flow control option and maximum amount of user data for received Network Data Messages are also specified here. Up to 16 bytes of optional user data can be passed to the other program.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-2%)	Send Connect Initiate Message subfunction code.
4	CHR$(*ula%*)	User link address, 1 to 255.
5–10	0%	Reserved - must be zero.
11	CHR$(*chnl%*)	Channel number of buffer containing the Connect Data Block and optional user data, 1 to 12. If 0, these are to be sent as part of the string.
12	CHR$(0%)	Reserved - must be zero.
13–14	*length%*	Length, in bytes, of information to be sent from the I/O buffer (including the Connect Data Block), 120 to 136. If 0, the number of bytes sent equals buffer size minus *offset*. Ignored if *chnl* = 0.
15–16	*offset%*	Offset, in bytes, from the beginning of the buffer where the Connect Data Block begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17–22	0%	Reserved - must be zero.
23	CHR$(*llmod%*)	Local link modifier. Indicates the type of flow control: 0 = none, 1 = segment, 2 = message.
24	CHR$(0%)	Reserved - must be zero.
25–26	*rmax%*	Maximum amount of user data (in bytes) that the local program will accept in one Network Data Message from the remote program.
27–40	0%	Reserved - must be zero.
41–160	*cdb$*	The Connect Data Block, if sent as part of the string (if *chnl* = 0). Ignored if *chnl* ≠ 0. (In either case, the Connect Data Block must = 120 bytes.)
161+	*data$*	Optional user data, if sent as part of the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned:

No meaningful data is returned.

Discussion:

ula

The user link address (*ula*) can be an integer from 1 to 255. This value will be used in later calls to refer to this link (assuming that the link being requested by this call is accepted by the remote NSP and the remote program).

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which channel buffer contains the Connect Data Block and the (optional) message data. Or it can be 0, indicating that this information is appended to the SYS call string.

length

The length parameter indicates how many bytes, including the Connect Data Block, are to be sent from the channel buffer. Since the Connect Data Block is 120 bytes long and the optional message data can be from 1 to 16 bytes long, acceptable values for *length* in this request run from 120 to 136; or it can be zero.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset* (which must still be in the range 120 to 136).

The *length* parameter is ignored if *chnl* = 0; that is, if the Connect Data Block and message data are appended to the SYS call string.

offset

If an I/O buffer is used, the Connect Data Block and message data are assumed to be in the buffer in contiguous bytes. They can be offset from the beginning of the buffer, and if so, the size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the Connect Data Block begins in byte 21:

byte 1 ... 20	21 ... 140	141
offset	Connect Data Block	user data

whole buffer

The *offset* field is ignored if *chnl* = 0.

llmod

The *llmod* value indicates the type of flow control requested for this end of the logical link.

Acceptable values and meanings are:

0 No flow control.
1 Segment flow control.
2 Message flow control.

Flow control is described in detail in Chapter 4. Remember that flow control is independent for each end of a logical link. What is requested here applies only to this end of the link. If *llmod* is nonzero, the calling program must issue Link Service calls to request Data Messages from the other program.

rmax

A program that has limited buffer space and is not designed to process large messages in small pieces (see the discussion of the *length* parameter for the Receive call, Section 5.7) can impose a limit on the amount of user message data it is willing to receive on a logical link. The *rmax* value specifies this limit in bytes.

The size of the receive buffers allocated by NSP will itself limit the amount of data that can be passed in a single Network Data Message. If the maximum size set for NSP by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local NSP will alter the maximum to the smaller limit before forwarding the Connect Initiate Message to the remote node. If $rmax = 0$, NSP will supply the size of its receive buffers as a default value. (The calling program will be informed of any such change by a field in the Connect Confirm Message from the target program when the link is accepted.)

The remote NSP uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS system, this transmit maximum is passed on to the remote program which must limit the amount of the user message data it transmits over the logical link according to this value.

connect data block

The Connect Data Block identifying the remote node and program can be specified as part of the SYS call string (*cdb*) or it can be given in the I/O buffer ($chnl \neq 0$). In either case, it must be 120 bytes long. The format of the 120-byte Connect Data Block is shown in Figure 5–2 and described in detail in Table 5–3.

data

Optional user data (up to 16 bytes) can be given following the Connect Data Block as part of the SYS call string (*data*) or following the Connect Data Block in the I/O buffer ($chnl \neq 0$).

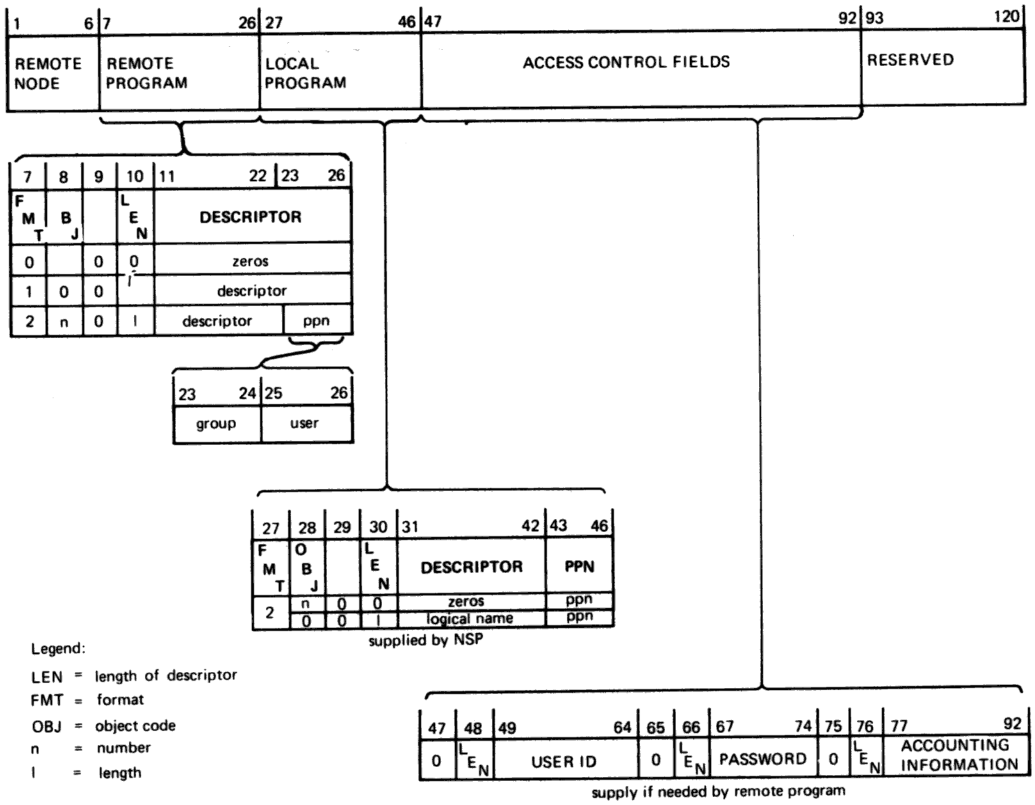


Figure 5–1: Format of Connect Data Block on Send Connect Initiate Message

Table 5–3: Format of Connect Data Block on Send Connect Initiate Message

Bytes	Content
1–6	**Remote Node.** 1 to 6 uppercase, alphanumeric characters giving the name of the remote node to which the Connect Initiate Message is to be sent. This name must contain at least one alphabetic character and names shorter than six characters must be left-justified and padded to six characters with spaces. If no node name is specified (that is, the request is directed to a program at the local node), this field should contain 6 spaces.
7–26	**Remote Program.** Identifies the remote program to which the Connect Initiate Message is directed as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of three formats:

Format 0

The remote program is identified by object type code alone. The fields are:

Bytes	Content
7	Zero, to indicate a format 0 name.
8	Object type code of the remote program.
9–26	Zero (not used).

(continued on next page)

Format 1

The remote program is identified by a descriptor. The fields are:

Bytes	Content
7	One, to indicate a format 1 name.
8	Normally zero. This byte should be zero unless the target DECnet system allows identification by both name and object type code on incoming connect requests (DECnet/E does not). If nonzero, this byte is interpreted as the object type code of the remote program for which the Connect Initiate Message is intended.
9	Reserved - must be zero.
10	Length of descriptor, in bytes, 0 to 16. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11–26).
11–26	Remote program descriptor (left-justified). For a DECnet/E program, this is the logical name that the remote program defined in its Declare Receiver system call. For other systems, it is the descriptor used to register the program for network operations.

Format 2

This format allows identification of the remote program by name or object type code and by the group and user codes under which the remote program is accessible on the remote system. (The group and user codes refer to what is called the project-programmer number on some DIGITAL systems, including RSTS/E. Other systems refer to these codes as the user identification code, or UIC.) DECnet/E systems use only the object type code or descriptor to find or start the target program for which a connect request is intended. The group and user codes are simply passed on to the target program for its own checking. Other DECnet systems can handle an incoming connect request in format 2 differently.

The subfields are defined as follows:

Bytes	Content
7	Two, to indicate a format 2 name.
8	Object type code of the remote program, if applicable. If zero, the descriptor must be used.
9	Reserved - must be zero.
10	Length of descriptor, 0 to 12. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
11–22	Remote program descriptor, if this is used instead of object type code. The descriptor should be left-justified in the field.
23–24	Remote group code. Binary value giving the group number under which the remote program is running or is to be started. Corresponds to the project number in a RSTS/E project-programmer number. This value is not used by DECnet/E systems but is simply passed on to the receiving program.
25–26	Remote user code. Binary value giving the user number under which the remote program is running or is to be started. Corresponds to the programmer number in a RSTS/E project-programmer number. This value is not used by DECnet/E systems but is simply passed on to the receiving program.

27–46 **Local Program**. This field is filled in by the local NSP using information that the local program specified in its Declare Receiver call. Any values passed by the user program in this field are ignored.

(continued on next page)

The information is passed to the remote system in format 2, as described previously. Byte 27 is filled in with a value of 2. If the program declared its identity with object type alone, the object type code is passed in byte 28 and bytes 30-42 are zero. If the program declared its identity with a logical name, or with a logical name and an object type code, the logical name is supplied in bytes 31-42 and byte 28 is zero. The length of the logical name is given in byte 30. The project-programmer number under which the program is running is passed in bytes 43-46 as the group and user codes. The project number is in byte 43. The programmer number is in byte 45. Both are in binary format.

47–92 **Access Control Fields**. These bytes can be used to define the calling program's access rights to the remote system or to the remote program's services. The idea is analogous to logging in on the remote system. The most rigorous checking done in a normal log-in on DIGITAL systems involves user identification, password, and account number, so three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all, nor does it require that the fields be used as described. A DECnet/E system simply passes this information onto the receiving program. To determine what, if anything, a non-RSTS DECnet system requires in these fields, see the DECnet manual for that system.

NOTE

The DECnet/E utilities NFT and NETCPY, which use logical links to access remote files and devices, use these fields to establish the local terminal user's right to access the remote files and devices. Both utilities prompt the local user for log-in information to be used at the remote system and pass that information along in the Connect Data Block of the Connect Initiate Message. The remote FAL or receiving NETCPY checks this information before allowing the operation requested.

Bytes	Content
47	Reserved – must be zero.
48	Length of user identification information specified in the next field, 0 to 16 bytes.
49–64	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the project-programmer number used at log-in. This information need only be specified here if it is required by the remote system or program. If it is not required, the length (byte 48) should be set to zero.
65	Reserved – must be zero.
66	Length of password information specified in the next field, 0 to 8 bytes.
67–74	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the project-programmer number at log-in. A password supplied here must be acceptable to the remote system or program, if one is required. If not required, the length (byte 66) should be set to zero.
75	Reserved – must be zero.
76	Length of the accounting information specified in the following field, 0 to 16 bytes.
77–92	Accounting Information. Many systems require a billing account number in addition to user identification and password. The accounting field is provided for this purpose. Once again, such information need only be specified here if it is required by the remote system or program. If it is not required, the length (byte 76) should be set to zero.

93–120 **Reserved for future use**. Currently ignored by DECnet/E but should be passed as zeros.

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
3	ACCOUNT OR DEVICE IN USE A logical link is currently active for the calling program with the same user link address (*ula*) as specified in this Connect Initiate Message. A different *ula* must be used or the existing link must be disconnected.
6	NOT A VALID DEVICE The node named in the Connect Data Block is not known to the system.
9	I/O CHANNEL NOT OPEN The channel specified by byte 11 of the call is not open. The program must OPEN the channel before using it for message operations.
14	DEVICE HUNG OR WRITE LOCKED The node named in the Connect Data Block is known to the system but is currently inactive. There is no physical communication path to the node.
17	TOO MANY OPEN FILES ON UNIT NSP cannot allocate its local link address (*lla*) for the logical link. The maximum number of logical links allowed at the local RSTS/E node (some limit between 1 and 63, as set by the system manager) has been reached. A later try may succeed. (See the system manager if the limit in effect is inadequate.)
18	ILLEGAL SYS() USAGE One of several possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The Declare Receiver call for the calling program had a null logical name and a zero object type code. There is no way to identify the program to the remote system. Do a Remove Receiver call and reissue the Declare Receiver call. 3. An invalid value was detected in one of the following fields: • Remote program field in the Connect Data Block. • One of the three access control fields in the Connect Data Block. • User link address (*ula*) given was zero. Valid values range from 1 to 255. • The *llmod* value (byte 23) must be 0, 1, or 2. • The remote descriptor length (byte 10 in the Connect Data Block) was nonzero, but the descriptor field was null (all zeros).
22	DISK PACK IS LOCKED OUT NSP cannot create a new logical link because the physical line to the remote node or the local node itself has been scheduled for shutdown by the system manager.

(continued on next page)

ERR Value	Error Text and Meaning
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13–16 are invalid. The following must be true: 1. The *offset* must be less than the buffer size. 2. The *length* must be less than or equal to the buffer size minus the offset. 3. The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 120 and 136. This encompasses 120 bytes for the Connect Data Block and a maximum of 16 bytes of data. The *offset* and *length* fields are checked for validity whenever *chnl* (byte 11) ≠ 0.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until NSP is enabled by the system manager.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

This example shows a Connect Initiate Message directed to a program with object type code 126 at node BOSTON. The Connect Data Block (CDB$) uses a format 0 name to address the remote program. The local program name is supplied by NSP and no access control fields are being used, so the rest of the Connect Data Block is filled with binary zeros to make up 120 bytes.

A user link address of 13 is used, segment flow control is requested by the calling program, and a receive maximum of 256 bytes is specified.

```
ULA% = 13%     !USER LINK ADDRESS = 13
LLMOD% = 1%    !SEGMENT FLOW CONTROL REQUESTED
RMAX% = 256% !RECEIVE MAXIMUM SET TO 256 BYTES
CDB$ = "BOSTON" + CHR$(0%) + CHR$(126%) + STRING$(112%,0%)
!
! EXAMPLE OF SEND CONNECT INITIATE
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-2%) + CHR$(ULA%)
        + STRING$(18%,0%) + CHR$(LLMOD%) + CHR$(0%)
        + CHR$(RMAX%) + CHR$(SWAP%(RMAX%))
        + STRING$(14%,0%) + CDB$ + "TIME-CARD UPDATE")
```

5.6.3 Send Connect Confirm Message (DECnet)

This system call is used to accept a remote program's request for a logical link. The call establishes the user link address that the local program will use to refer to the link and identifies the link being accepted by the local link address assigned by the local NSP. The flow control option and receive maximum are specified here and up to 16 bytes of optional message data can be passed to the remote program.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-3%)	Send Connect Confirm Message subfunction code.
4	CHR$(*ula%*)	User link address, 1 to 255.
5–6	*lla%*	Local link address (from bytes 5–6 of received Connect Initiate Message). (See Section 5.7.2.)
7–10	0%	Reserved – must be zero.
11	CHR$(*chnl%*)	Channel number of buffer containing the optional message data, 1 to 12. If 0, the data is appended to the SYS call string.
12	CHR$(0%)	Reserved – must be zero.
13–14	*length%*	Length, in bytes, of data to be sent from the buffer, 1 to 16. If 0, the number of bytes sent equals buffer size minus *offset*. Ignored if *chnl* = 0.
15–16	*offset%*	Offset, in bytes, from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17–22	0%	Reserved – must be zero.
23	CHR$(*llmod%*)	Local link modifier, used to indicate the type of flow control: 0 = none, 1 = segment, 2 = message.
24	CHR$(0%)	Reserved – must be zero.
25–26	*rmax%*	Maximum amount of user data (bytes) that the local program will accept in one Network Data Message from the remote program.
27–40	0%	Reserved – must be zero.
41+	*data$*	Optional message data, if appended to the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned:

No meaningful data is returned.

Discussion:

ula

The user link address (*ula*) can be a number from 1 to 255. This value is used in later calls to refer to this link. The *ula* must be unique. That is, one program cannot have two logical links with the same *ula* at the same time.

lla

The local link address (*lla*) identifies the link being accepted. It is a number assigned by the local NSP when it received the Connect Initiate Message requesting the link. The local link address is passed to the local program in bytes 5–6 of the received Connect Initiate Message to which this Connect Confirm Message is a positive response. (Section 5.7.2 shows the entire format of a received Connect Initiate Message.)

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which buffer contains the (optional) user data. Or it can be 0, indicating that this information is appended to the SYS call string.

length

The *length* parameter indicates how many bytes are to be sent from the channel buffer. Acceptable values for *length* in this request can range from 1 to 16, or it can be zero.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This size must also be in the range 1 to 16.

The *length* parameter is ignored if *chnl* = 0; that is, if the user data is appended to the SYS call string.

offset

If an I/O buffer is used, the user data is assumed to be in the buffer in contiguous bytes. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:

byte 1 ... 20	21 ...
offset	user data

whole buffer

The *offset* field is ignored if *chnl* = 0.

llmod

The *llmod* value indicates the type of flow control requested for this end of the logical link. Acceptable values and meanings are:

0 No flow control.
1 Segment flow control.
2 Message flow control.

Flow control is described in detail in Chapter 4. Remember that flow control is independent for each end of a logical link — what is requested here applies only to this end of the link. If *llmod* is nonzero, the calling program must issue Link Service calls to request Network Data Messages from the remote program.

rmax

A program that has limited buffer space and is not designed to process large messages in small pieces (see the discussion of the *length* parameter for the Receive call, Section 5.7) can impose a limit on the amount of user data it is willing to receive in one Network Data Message from the remote program over this logical link. The *rmax* value specifies this limit in bytes.

The size of the receive buffers allocated by NSP will itself limit the amount of data that can be passed in a single Network Data Message. If the maximum size set for NSP by Network Management is not large enough to handle the receive maximum as specified by *rmax*, the local NSP will alter the maximum to the smaller limit before forwarding the Connect Confirm Message to the remote node. If $rmax = 0$, NSP will supply the size of its receive buffers as a default.

The remote NSP uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS system, this transmit maximum is passed on to the remote program which must limit the amount of the user data it transmits in Network Data Messages according to this value.

data

Up to 16 bytes of optional user data can be passed as part of the SYS call string (*data*) or in the channel buffer ($chnl \neq 0$).

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
3	ACCOUNT OR DEVICE IN USE A logical link is currently active for the calling program with the same user link address (*ula*) as specified in this Connect Confirm Message. A different *ula* must be used or the existing link must be disconnected.
5	CAN'T FIND FILE OR ACCOUNT The local link address (*lla*) specified in bytes 5–6 does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating Connect Initiate Message has been cancelled. In the latter case, a Link Abort Message has already been queued for this link. Since no *ula* was ever established, an *lla* field is available in a received Link Abort Message identifying the link.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 is not open. The program must OPEN the channel before using it for message operations. See Section 5.1.2.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Connect Confirm Message for a link that is not waiting for confirmation or rejection will result in this error. For example, a second Connect Confirm Message for the same logical link will result in this error because the link is already established.

(continued on next page)

ERR Value	Error Text and Meaning
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The program has not issued a Declare Receiver call. All DECnet functions require the caller to be a declared receiver. 2. An invalid value was detected in one of the following fields: • *ula* (byte 4) is 0. Acceptable values range from 1 to 255. • *llmod* (byte 23) must be 0, 1, or 2.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13–16 are invalid. The following must be true: 1. The *offset* must be less than the buffer size. 2. The *length* must be less than or equal to the buffer size minus the *offset*. 3. The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 0 and 16.
32	BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until NSP is enabled by the system manager.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

In this example, a Send Connect Confirm Message system call is issued in response to a previously received Connect Initiate Message whose control information was returned to the target string A$. The MID function places the local link address from A$ in the string LLA$. Other parameters are similar to those in a Connect Initiate Message.

```
LLA$ = MID(A$,5%,2%)      !GET LOCAL LINK ADDRESS
ULA% = 25%                !USER LINK ADDRESS = 25
LLMOD% = 2%               !MESSAGE FLOW CONTROL REQUESTED
RMAX% = 512%
!
! EXAMPLE OF CONNECT CONFIRM
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-3%) + CHR$(ULA%)
        + LLA$ + STRING$(16%,0%)
        + CHR$(LLMOD%) + CHR$(0%)
        + CHR$(RMAX%) + CHR$(SWAP%(RMAX%)))
```

5.6.4 Send Connect Reject Message (DECnet)

Sending a Connect Reject Message rejects a remote program's request for a logical link. The link is identified by the local link address established by the local NSP. Up to 16 bytes of optional user data can also be passed to the remote program.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-4%)	Send Connect Reject Message subfunction code.
4	CHR(0%)	Ignored.
5–6	*lla%*	Local link address (from bytes 5–6 of received Connect Initiate Message).
7–10	0%	Reserved - must be zero.
11	CHR$(*chnl%*)	Channel number of buffer containing optional message data, 1 to 12. If 0, data is appended to SYS call string.
12	0%	Reserved - must be zero.
13–14	*length%*	Length, in bytes, of data to be sent from the buffer, 0 to 16. If 0, the number of bytes sent equals the buffer size minus *offset*. Ignored if *chnl* = 0.
15–16	*offset%*	Offset (in bytes) from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17–22	0%	Reserved - must be zero.
23–24	*reason%*	Reason for rejection, if applicable. Can be 0 (normal case for user programs), 34 (access not permitted), or 36 (invalid accounting information).
25–40	0%	Reserved - must be zero.
41–56	*data$*	Optional user data, if appended to the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned:

No meaningful data is returned.

Discussion:

lla

The local link address (*lla*) identifies the link being rejected. It is a number assigned by the local NSP when it received the Connect Initiate Message requesting the link. The *lla* is passed to the local program in bytes 5–6 of the received Connect Initiate Message to which this Connect Reject Message is a negative response. (Section 5.7.2 shows the entire format of a received Connect Initiate Message.)

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which buffer contains the (optional) message data to be passed to the remote program. Or it can be 0, indicating that this information is appended to the SYS call string.

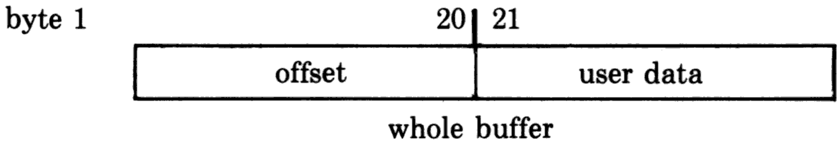
length

The *length* parameter indicates how many bytes of user data, if any, are to be sent from the channel buffer. Acceptable values range from 1 through 16. This field is ignored if *chnl* = 0.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This size must also be in the range 1 to 16.

offset

If a channel buffer is used, the user message data, if any, can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

reason

DECnet/E NSP allows a program to specify one of the following three reason codes: 0, 34, or 36.

Most user programs will simply use 0. If any reason is supplied for rejecting the connection, it is passed in the 16-byte user data area.

The values 34 and 36 are allowed so that DECnet/E support programs (such as FAL and NFT) can reject unauthorized or improper requests for access to local RSTS/E files in a manner agreed upon as part of the overall DECnet design. There is no reason for a user program to use these codes unless it talks to other programs that interpret them. All DECnet programs that are concerned with protecting against unauthorized access are designed to recognize these codes and take appropriate action, such as displaying an error message to a user, when they receive such a connect rejection.

The value 34 indicates that the user ID and password subfields of the access control field in the Connect Data Block in the remote program's Connect Initiate Message do not correspond to any valid user known to the local system.

The value 36 indicates that, while the user ID and password subfields are acceptable, the account subfield is not. For example, the specified user may not be authorized to use that billing account or the account may be exhausted.

data

Up to 16 bytes of optional user data can be appended to the SYS call string (*data*) or sent from the I/O buffer (*chnl* $\neq 0$).

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
5	CAN'T FIND FILE OR ACCOUNT The local link address (*lla*) given in bytes 5–6 does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating Connect Initiate request has been cancelled. In the latter case, a Disconnect or Link Abort Message has already been queued for this link. Since no *ula* was ever established, an *lla* field is available in the received Link Abort Message identifying the link.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 is not open. The program must OPEN the channel buffer before using it for message operations. See Section 5.1.2.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Connect Reject Message for a logical link that is not waiting for confirmation or rejection will cause this error. For example, a Connect Reject Message for a link that is already established will return this error.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program has not issued a Declare Receiver call. All DECnet functions require the caller to be a declared receiver. 2. The *reason* code specified in bytes 23–24 was not 0, 34, or 36.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13–16 are inconsistent. The following relationships must be true: 1. The *offset* must be less than the buffer size. 2. The *length* must be less than or equal to the buffer size minus the *offset*. 3. The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 1 and 16 for a Send Connect Reject Message call.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store the user data portion of the Connect Reject Message. A retry may succeed.

(continued on next page)

ERR Value	Error Text and Meaning
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until NSP is enabled by the system manager.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

In this example, a Connect Reject Message is sent to a remote program in response to a received Connect Initiate Message whose control information was returned to the target string B$.

```
!
!EXAMPLE OF CONNECT REJECT
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-4%) + CHR$(0%)
        + MID(B$,5%,2%) + STRING$(34%,0%)
        + "TOO LATE")
```

5.6.5 Send Network Data Message (DECnet)

The Send Network Data Message call is used to transmit user data — a *segment*, as defined in Chapter 4 — over an established logical link. A flag can be set to indicate that the segment is also the end of a logical message. If the remote program requested flow control when the link was established, requests for data must be outstanding before this call can be issued.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-5%)	Send Network Data Message subfunction code.
4	CHR$(*ula%*)	User link address.
5–10	0%	Reserved - must be zero.
11	CHR$(*chnl%*)	Channel number of buffer containing the user message data, 1 to 12. If 0, this data is appended to the SYS call string.
12	CHR$(0%)	Reserved - must be zero.
13–14	*length%*	Length of data to be sent from the buffer (bytes). If 0, the number of bytes sent equals the buffer size minus *offset*. Ignored if *chnl* = 0.
15–16	*offset%*	Offset (in bytes) from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17–20	0%	Reserved - must be zero.
21	CHR$(*dmflgs%*)	Data message flags: *rls%* + *eom%* + *bom%*. *rls* = 128, link status data returned *rls* = 0, no link status data returned **eom bom** 0 0 Middle segment of a message 0 1 First segment of a message 2 0 Last segment of a message 2 1 Sole segment of a message
22–40	0%	Reserved - must be zero.
41+	*data$*	User data, if appended to the string.

Data Returned:

If *rls* = 128, link status information is returned when this call is successfully completed. The data returned is identical in format to a received Link Service Message, as described in Section 5.7.7.

Discussion:

ula

The user link address (*ula*) is the link identifier (1 to 255) chosen by the calling program in its Connect Initiate Message or Connect Confirm Message sent during the connection sequence.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which I/O buffer contains the user message data to be sent to the remote program. Or it can be 0, indicating that this information is appended to the SYS call string.

length

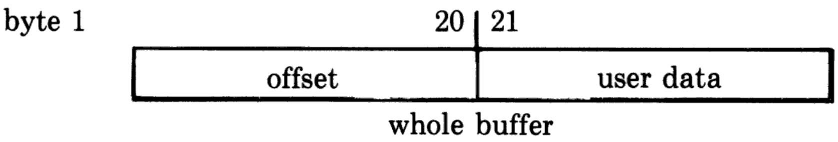
The *length* parameter indicates how many bytes are to be sent from the I/O buffer. The maximum amount of data that can be transferred over this link is established by the remote program or remote NSP. It is passed on to the local program as a transmit maximum (the *tmax* parameter in a received Connect Initiate or Connect Confirm Message, Sections 5.7.2 and 5.7.3). Thus, *length* can be any value from one through *tmax*.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This length is still limited by the transmit maximum, as described previously.

The *length* parameter is ignored if *chnl* = 0; that is, if the user data is appended to the SYS call string.

offset

If an I/O buffer is used, the user message data can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

dmflgs

The *dmflgs* field can be coded as the sum of three bit values:

dmflgs% = rls% + eom% + bom%

The low-order two bits of the *dmflgs* parameter (*eom% + bom%*) indicate whether the segment being sent in this Network Data Message is the first, last, middle, or sole segment of a logical message. If message flow control was

requested by the remote program when the link was established, NSP uses these two bits to regulate the transmission of logical messages as the remote program requests them. (See Chapter 4 for a full discussion of flow control.)

eom	*bom*	
0	0	Middle segment of a logical message
0	1	First segment of a logical message
2	0	Last segment of a logical message
2	1	Sole segment of a logical message

DECnet/E NSP uses the *eom* bit to count logical messages transmitted to the remote program. The *bom* bit is not used in maintaining the logical message request count but is simply passed to the remote system and program.

NSP also uses the *eom* bit to determine whether it will permit a disconnect of the logical link. If a disconnect is requested but the last Network Data Message sent did not have the *eom* bit set, NSP assumes that the last message has not been completely sent and will not permit the disconnect. This is always done, irrespective of the flow control option selected by the remote program.

Some DECnet implementations require that the *eom* flag be set to terminate an asynchronous I/O operation. Thus, when the remote program requests segment flow control or no flow control, the *eom* flag should always be set. Setting the flag in this case will not affect a remote DECnet/E node but is essential if the remote node is a DECnet node waiting to terminate asynchronous I/O.

The sign bit of this byte (*rls*) is used to request that link status information be returned to the target string when the send call has successfully completed:

rls = 128 Link status information is returned.
rls = 0 No link status information is returned.

If link status information is requested, it is returned in the same format as a received Link Service Message, as described in Section 5.7.7.

data

The user data to be transmitted to the remote program can be appended to the SYS call string (*data*) or sent from the I/O buffer (*chnl* $\neq$ 0). In either case, the amount of data that can be transmitted is limited by the remote program or remote NSP. This transmit maximum is passed to the calling program in the *tmax* parameter in the received Connect Initiate or Connect Confirm Message (Section 5.7.2 or 5.7.3) for this logical link.

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
4	NO ROOM FOR USER ON DEVICE The Data Message transmit queue for this logical link is full. NSP is waiting for acknowledgment from the remote system for previously sent Network Data Messages. No more can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should simply retry the send after a short (1 to 5 second) delay. (See Chapter 4 for a full discussion of flow control.)
5	CAN'T FIND FILE OR ACCOUNT The user link address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
9	I/O CHANNEL NOT OPEN The I/O channel specified in byte 11 is not open. The program must OPEN the buffer before using it for message operations. See Section 5.1.2.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Network Data Message over a logical link that has not yet been confirmed will cause this error.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program has not issued a Declare Receiver call. All DECnet functions require the caller to be a declared receiver. 2. The *ula* specified is zero. It must be within the range 1 to 255.
19	DISK BLOCK IS INTERLOCKED Either the remote system has inhibited transmission on this link because of a backpressure condition or there are no outstanding requests for segments or logical messages from the remote program (assuming it requested segment or message flow control for the link). The program will be notified with a Link Service Message when the condition clears, as described in Chapter 4.

(continued on next page)

ERR Value	Error Text and Meaning
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13–16 are inconsistent. The following relationships must be true: 1. The *offset* must be less than the buffer size. 2. The *length* must be less than or equal to the buffer size minus the *offset*. 3. The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 1 and the transmit maximum (*tmax*) established by the remote program or system in its Connect Initiate or Connect Confirm Message for this logical link.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until NSP is enabled by the system manager.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following Send Network Data call sends user data from a buffer. The null device is opened on channel 2 to obtain buffer space, the message data is loaded into the buffer using LSET, and the user data is sent. The use of JUNK$ at the beginning of the buffer illustrates the use of the buffer offset field.

```
ULA% = 14%          !USER LINK ADDRESS = 14
DMFLGS% = 1% + 2%
!SOLE SEGMENT OF LOGICAL MESSAGE
CHNL% = 2%          !I/O BUFFER ON CHANNEL 2
OPEN "NL:" AS FILE CHNL%, RECORDSIZE 100%
FIELD CHNL%, 10% AS JUNK$, 90% AS TEXT$
MSG$ = :"THIS MESSAGE WAS SENT FROM A BUFFER"
LENGTH% = LEN(MSG$)
OFFSET% = LEN(JUNK$)
LSET TEXT$ = MSG$
!
!EXAMPLE OF SEND NETWORK DATA MESSAGE
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-5%) + CHR$(ULA%)
        + STRING$(6%,0%) + CHR$(CHNL%) + CHR$(0%)
        + CHR$(LENGTH%) + CHR$(SWAP%(LENGTH%))
        + CHR$(OFFSET%) + CHR$(SWAP%(OFFSET%))
        + STRING$(4%,0%) + CHR$(DMFLGS%))
```

5.6.6 Send Interrupt Message (DECnet)

The Send Interrupt Message call transmits an Interrupt Message to a remote program over an established logical link. All DECnet systems are designed to deliver Interrupt Messages ahead of other messages. If the remote system is a DECnet/E system, the Interrupt Message will be placed at the head of the pending message queue, behind the first message (since it can already be partly received) and behind any other pending Interrupt Messages queued for the program. Interrupt Messages are subject to flow control, as described in Chapter 4.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-6%)	Send Network Interrupt Message subfunction code.
4	CHR$(*ula*%)	User link address.
5–10	0%	Reserved - must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the user message data, 1 to 12. If 0, this data is to be appended to the SYS call string.
12	CHR$(0%)	Reserved - must be zero.
13–14	*length*%	Length, in bytes of data to be sent from the buffer, 1 to 16. If 0, the number of bytes sent equals the buffer size minus *offset*. Ignored if *chnl* = 0.
15–16	*offset*%	Offset (in bytes) from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17–20	0%	Reserved - must be zero if passed.
21	*inflgs*%	Interrupt flags. *inflgs*% = *rls*% *rls* = 128, link status data returned. *rls* = 0, no link status data returned.
22–40	0%	Reserved - must be zero if passed.
41–56	*data*$	Optional user data, if appended to the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned:

If *rls* = 128, link status information is returned when this call is successfully completed. The data returned is identical to a received Link Service Message, as described in Section 5.7.7.

Discussion:

ula

The user link address (*ula*) is the link identifier (1 to 255) chosen by the calling program in its Connect Initiate Message or Connect Confirm Message sent during the connection sequence for this logical link.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which I/O buffer contains the user data to be sent to the remote program. Or it can be 0, indicating that this information is appended to the SYS call string.

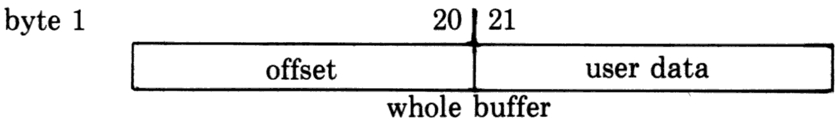
length

The *length* parameter indicates how many bytes are to be sent from the I/O buffer. Acceptable values range from 1 through 16. This field is ignored if *chnl* = 0.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the channel. That is, the number of bytes sent is equal to the buffer size minus *offset*. This length is also limited to 1 to 16 bytes.

offset

If an I/O buffer is used, the user data, if any, can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

inflgs

The *inflgs* parameter indicates whether link status information is to be returned to the target string when this call is successfully completed. This field should be coded as follows:

inflgs% = *rls%*

where

rls = 128 Link status information is returned.
rls = 0 No link status information is returned.

If link status is requested, it is returned in the same format as a received Link Service Message (Section 5.7.7).

data

Up to 16 bytes of optional user data can be appended to the SYS call string (*data*) or sent from the I/O buffer (*chnl* $\neq$ 0).

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. NSP is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Message can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should simply retry the send after a short delay of from 1 to 5 seconds. (See Chapter 4 for a full discussion of flow control.)
5	CAN'T FIND FILE OR ACCOUNT The user link address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 is not open. The program must OPEN the I/O buffer before using it for message operations. See Section 5.1.2.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending an Interrupt Message over a logical link that has not yet been confirmed will cause this error to occur.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program has not issued a Declare Receiver call. All DECnet functions require the caller to be a declared receiver. 2. The *ula* specified was zero. It must be within the range 1 to 255.
19	DISK BLOCK IS INTERLOCKED The remote NSP has prohibited transmission of Interrupt Messages over this logical link according to its rules for Interrupt Message flow control. (See Chapter 4 for a full discussion of flow control.) A Link Service Message will be delivered to the program when the condition clears, as described in Chapter 4.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13–16 are inconsistent. The following relationships must be true: 1. The *offset* must be less than the buffer size. 2. The *length* must be less than or equal to the buffer size minus the *offset*. 3. The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 1 and 16 for an Interrupt Message.

(continued on next page)

ERR Value	Error Text and Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until NSP has been enabled by the system manager.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example illustrates a Send Interrupt Message call. The user data, START PHASE 2, is passed to a remote program over logical link 170. (The remote program would have to be coded to recognize this message and take appropriate action.)

```
ULA% = 170%
!
!EXAMPLE OF SEND INTERRUPT CALL
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-6%) + CHR$(ULA%)
        +STRING$(36%,0%) + "START PHASE 2")
```

5.6.7 Send Link Service Message (DECnet)

The Send Link Service Message call is used in three ways: (1) to request data from a remote program over a flow-controlled logical link (Chapter 4); (2) to reenable incoming Interrupt Messages over a logical link (Chapter 4); or (3) to obtain status information for the link. No user data is allowed with a Link Service Message.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-7%)	Send Link Service Message subfunction code.
4	CHR$(*ula*%)	User link address.
5–12	0%	Reserved - must be zero.
13–16	0%	Ignored.
17–20	0%	Reserved - must be zero.
21	CHR$(*lsflgs*%)	Link service flags. *lsflgs*% = *rls*% + *opr*%. *opr*% indicates the purpose of the call: *opr* = 0 Request data from remote program over flow-controlled link = 1 Reenable Interrupt Messages = 2 Obtain status information If *rls* = 128, link status information is returned when *opr* = 0 or 1. If *rls* = 0, no link status information is returned when *opr* = 0 or 1.
22	CHR$(0%)	Reserved - must be zero.
23	CHR$(*drcnt*%)	Data request count. Increment for logical message or counter. Must be positive and not cause counter to exceed 127. Ignored when *opr* ≠ 0.
24	CHR$(*ircnt*%)	Interrupt request count. Must equal 1 when *opr* = 1. Ignored otherwise.
25–40	0%	Reserved - must be zero if passed.

Data Returned:

If *opr* = 2, or if *rls* = 128 when *opr* = 0 or 1, link status information is returned when this call completes successfully. The data returned is identical to a received Link Service Message, as described in Section 5.7.7.

Discussion:

ula

The user link address (*ula*) is the link identifier (1 to 255) chosen by the calling program in its Connect Initiate Message or Connect Confirm Message sent during the connection sequence for this logical link.

lsflgs

The link service flags byte is coded as the sum of two values: *lsflgs%* = *rls%* + *opr%*. The low-order two bits (*opr*) indicate the purpose of the call.

opr = 0 Indicates the call is requesting segments or logical messages from the remote program. In this case, the *drcnt* parameter is interpreted as the number of segments or logical messages being requested. This form of the call can be used only when the calling program requested segment or message flow control in its Connect Initiate or Connect Confirm Message for this logical link. If no flow control was requested, a Send Link Service call with *opr* = 0 will fail with ERR = 18.

= 1 Indicates the call is to reenable incoming Interrupt Messages for this link. In this case, the *ircnt* parameter is interpreted as the number of Interrupt Messages allowed (always 1).

= 2 Indicates the call is to obtain status information only. In this case, the *drcnt* and *ircnt* parameters are ignored.

The sign bit of this byte (*rls*) can be used when *opr* = 0 or 1 to request that link status information be returned to the SYS call string when the call has completed successfully.

rls = 128 Link status information is returned.
rls = 0 No link status information is returned.

Link status information is always returned when *opr* = 2. If status information is requested, it is returned in the same format as a received Link Service Message (Section 5.7.7).

drcnt

The data request count (*drcnt*) is meaningful only when *opr* = 0 and indicates the maximum number of segments or logical messages being requested from the remote program. Depending on the flow control option in effect for the logical link, the value is interpretted as either a segment request count (segment flow control) or as a logical message request count (message flow control). In either case, it is an incremental count, added to the counter of outstanding requests for segments or logical messages kept by the remote and local NSP (see Chapter 4). An error is returned if the *drcnt* given increases the counter to more than 127.

ircnt

The interrupt request count (*ircnt*) is meaningful only when *opr* = 1. It indicates the number of Interrupt Messages being requested. DECnet/E allows only one interrupt request outstanding for a logical link at any given time. Hence, *ircnt* must always be 1 when *opr* = 1 and has the effect of reenabling incoming interrupts on the logical link. (Remember that interrupts can be reenabled only when the previous Interrupt Message for this logical link, if any, has been received from the pending message queue. If it has not, the call will fail with ERR = 19.)

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. NSP is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and will not cause a Link Service Message to be queued when the condition clears. The program should simply retry the send after a short (1 to 5 second) delay. (See Chapter 4 for a full discussion of flow control.)
5	CAN'T FIND FILE OR ACCOUNT The user link address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
9	I/O CHANNEL NOT OPEN Since the Link Service Message does not provide for transmitting user message data, the byte used for channel number in all other system calls (byte 11) should always be zero.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Link Service Message over a logical link that has not yet been confirmed will cause this error.
18	ILLEGAL SYS() USAGE One of several possibilities: 1. The calling program has not issued a Declare Receiver call. All DECnet functions require the caller to be a declared receiver. 2. The *opr* value in byte 21 is not 0, 1, or 2. 3. The *opr* value is 0, but the calling program did not request flow control for the logical link in its Connect Initiate or Connect Confirm Message. 4. The data request count (*drcnt*) value given increased the count of outstanding requests to a value greater than 127. 5. The *opr* value is 1, indicating an Interrupt Message is being requested, but the *ircnt* parameter is not equal to 1. 6. The *opr* value is 1, but the outstanding requests counter for interrupts is already equal to 1. 7. The *ula* given is 0. It must be within the range 1 to 255.

(continued on next page)

ERR Value	Error Text and Meaning
19	DISK BLOCK IS INTERLOCKED The *opr* value is 1, indicating a request for an Interrupt Message, but there is already an Interrupt Message queued for this logical link. (See Chapter 4 for details on flow control.)
31	ILLEGAL BYTE COUNT FOR I/O This error can occur if byte 11 (the *chnl* number in all other system calls) is nonzero. In this case, the system attempts its consistency checking for length and offset fields in bytes 13–16 and has found them to be inconsistent. Since no message data can be transmitted with a Link Status Message, byte 11 should be zero.
32	NO BUFFER SPACE AVAILABLE System buffers are unavailable to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet system calls cannot be executed until NSP is enabled by the system manager.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example sends a Link Service Message to request five segments from the remote program over logical link 12. (Assume that segment flow control was requested by the program in its Connect Initiate or Connect Confirm Message that established the logical link.)

```
ULA% = 12%
LSFLGS% = 0%
DRCNT% = 5%
!
!EXAMPLE OF SEND LINK SERVICE MESSAGE CALL
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-7%) + CHR$(ULA%)
        +STRING$(16%,0%)+CHR$(LSFLGS%)+CHR$(0%)
        +CHR$(DRCNT%))
```

5.6.8 Send Disconnect Message (DECnet)

This call is used to terminate an established logical link. Messages in the pending message queue will remain. No more messages can be sent over the link, however. The user link address is freed and can be used for another logical link. If the last Network Data Message sent did not have the end-of-message flag set, or if any Network Data, Interrupt, or Link Service Messages are still waiting for acknowledgment in the transmit queues for this logical link, this call will fail with an error.

Successful transmission of a Disconnect Message assures the sender that the remote system has received and acknowledged all Network Data, Interrupt, and Link Service Messages previously sent over the link. It does not, however, guarantee that the receiving program has processed the messages. The Disconnect Message is useful in "master-slave" communication where the master program only transmits data and the slave program only receives data (see Section 3.13). Otherwise, the Disconnect Message provides no particular advantage over a Link Abort Message (Section 5.6.9) in terminating a logical link.

Up to 16 bytes of user data can be sent with the Disconnect Message.

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive function code.
3	CHR$(-8%)	Send Disconnect Message subfunction code.
4	CHR$(*ula*%)	User link address.
5–10	0%	Reserved - must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the (optional) message data, 1 to 12. If 0, data is appended to the string.
12	CHR$(0%)	Reserved - must be zero.
13–14	*length*%	Length, in bytes, of data to be sent from the buffer, 1 to 16. If 0, the number of bytes sent equals buffer size minus *offset*. Ignored if *chnl* = 0.
15–16	*offset*%	Offset (in bytes) from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17–40	0%	Reserved - must be zero.
41–56	*data*$	Optional user data, if appended to the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned:

No meaningful data is returned.

Discussion:

ula

The user link address (*ula*) is the number (1 to 255) that was established by the calling program in its Connect Initiate or Connect Confirm Message sent during the connection sequence for this logical link.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which I/O buffer contains the user data to be sent to the remote program. Or it can be 0, indicating that this information is appended to the SYS call string.

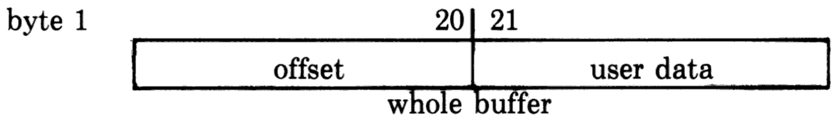
length

The *length* parameter indicates how many bytes are to be sent from the I/O buffer. Acceptable values range from 1 through 16. This field is ignored if *chnl* = 0.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the channel. That is, the number of bytes sent is equal to the buffer size minus *offset*. This length must also fall in the range of 1 to 16 bytes.

offset

If an I/O buffer is used, the user data, if any, can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

data

Up to 16 bytes of optional message data can be appended to the SYS call string (*data*) or sent from the channel buffer (*chnl* ≠ 0).

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
4	NO ROOM FOR USER ON DEVICE There are outstanding unacknowledged messages on either the Data Message transmit queue or the Interrupt/Link Service transmit queue. All messages previously sent over the logical link must be acknowledged before the Disconnect Message can be sent. (If an immediate unconditional termination of the logical link is desired, a Link Abort Message should be sent.)

(continued on next page)

ERR Value	Error Text and Meaning
5	CAN'T FIND FILE OR ACCOUNT The user link address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
9	I/O CHANNEL NOT OPEN The *chnl* specified in byte 11 is not open. The program must OPEN the channel buffer before using it for message operations. See Section 5.1.2.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Disconnect Message over a logical link that has not been confirmed will cause this error to occur.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program has not issued a Declare Receiver call. All DECnet functions require the caller to be a declared receiver. 2. The *ula* specified is 0. It must be within the range 1 to 255.
19	DISK BLOCK IS INTERLOCKED The last segment sent by the local program over this logical link did not have the *eom* flag set. NSP will not permit a logical link to be disconnected unless all messages have been completely sent. (Note that this test of the *eom* flag is done irregardless of the flow control option chosen by the remote program.)
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13–16 are inconsistent. The following must be true: 1. The *offset* must be less than the buffer size. 2. The *length* must be less than or equal to the buffer size minus the *offset*. 3. The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 1 and 16 for a Disconnect Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store the optional message data portion of the Disconnect Message. A later retry may succeed. (A request to send a Disconnect Message will never fail for this reason if no user data is passed.)
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until NSP is enabled by the system manager.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example disconnects logical link 230 and includes user data explaining the reason for the disconnect. (The remote program would have to be coded to recognize this message and take appropriate action.)

```
!
!EXAMPLE OF SEND DISCONNECT
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-8%) + CHR$(230%) +
        STRING$(36%,0%) + "END-OF-FILE")
```

5.6.9 Send Link Abort Message

This call is used to terminate a logical link immediately. Unlike the Disconnect Message, no attempt is made to ensure that all messages previously sent over this logical link have been received by the remote system. If there are still messages in the transmit queues waiting for acknowledgment from the remote NSP, they will be discarded. They may or may not have been received by the remote system (see Section 3.14).

Up to 16 bytes of user data can be specified in the message. If the Link Abort Message is issued for an established logical link, the remote system is notified that the link has been broken and the user data is delivered to the remote program. If the Link Abort Message is issued for a logical link awaiting confirmation from the remote program, the user data does not reach the remote program (see Section 3.14).

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/receive subfunction code.
3	CHR$(-9%)	Send Link Abort Message subfunction code.
4	CHR$(*ula*%)	User link address.
5–10	0%	Reserved - must be zero.
11	*chnl*%	Channel number of buffer containing the (optional) user data, 1 to 12. If 0, this data is to be appended to the string.
12	0%	Reserved - must be zero.
13–14	*length*%	Length, in bytes, of data to be sent from the buffer, 1 to 16. If 0, the length is equal to the buffer size minus *offset*. Ignored if *channel* = 0.
15–16	*offset*%	Offset (in bytes) from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17–40	0%	Reserved - must be zero.
41+	*data$*	Optional user data, if appended to the string; maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned:

No meaningful data is returned.

Discussion:

ula

The user link address (*ula*) is the number (1 to 255) that was established by the calling program in its Connect Initiate or Connect Confirm Message sent during the connection sequence for this logical link.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which I/O buffer contains the user data to be sent to the remote program. Or it can be 0, to indicate that this information is appended to the SYS call string.

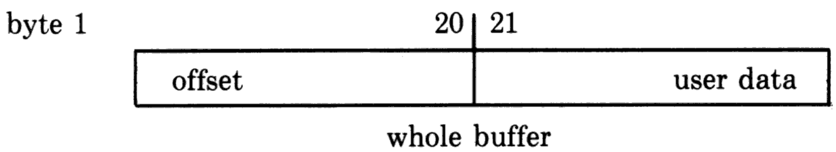
length

The *length* parameter indicates how many bytes are to be sent from the I/O buffer. Acceptable values range from 1 through 16. Or, it can be zero.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This length must also fall in the range of from 1 to 16 bytes.

offset

If a channel buffer is used, the user message data, if any, can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

data

Up to 16 bytes of optional user data can be appended to the SYS call string (*data*) or sent from the channel buffer (*chnl* ≠ 0). This data will not reach the remote program if the Link Abort Message is issued for a link awaiting confirmation from the remote program (see Section 3.14).

Possible Errors:

ERR Value	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
5	CAN'T FIND FILE OR ACCOUNT The user link address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.

(continued on next page)

ERR Value	Error Text and Meaning
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 is not open. The program must OPEN the I/O buffer before using it for message operations. See Section 5.1.2.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program has not issued a Declare Receiver call. All DECnet functions require the caller to be a declared receiver. 2. The *ula* specified is 0. It must be within the range 1 to 255.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13–16 are inconsistent. The following must be true: 1. The *offset* must be less than the buffer size. 2. The *length* must be less than or equal to the buffer size minus the offset. 3. The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 1 and 16 for a Link Abort Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later retry may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until NSP is enabled by the system manager.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example aborts logical link 31.

```
!
!EXAMPLE OF SEND LINK ABORT MESSAGE
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-9%) + CHR$(31%))
```

5.7 Receive System Call

The Receive call is used to retrieve a message from the calling program's pending message queue. It returns control parameters to the target string and user data, if any, to an I/O buffer specified in the call. The Connect Data Block for Connect Initiate Messages is also delivered to the I/O buffer. The user data can be retrieved all at once or in portions, and portions can be discarded.

A Receive call can be selected to get (1) the first message in the queue, (2) the first message in the queue from either a network or a local sender, or (3) the first message in the queue for a particular logical link. A "sleep" flag and associated timer can be set to indicate that the calling program is to wait if there are no appropriate messages pending in the queue. Otherwise, if there are no appropriate messages pending, the Receive call terminates with an error (ERR = 5).

Data Passed:

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR(22%)	Send/receive function code.
3	CHR$(2%)	Receive subfunction code.
4	CHR$(*rmod%*)	Receive modifier: *n%+l%+t%+s%* *s* Sleep if no messages? 0 = no, 1 = yes *t* Truncate excess data? 0 = no, 2 = yes *l* Select local messages? 0 = no, 4 = yes *n* Select network messages? 0 = no, 8 = yes (*n* ignored if *l* = 4) If *l* and *n* = 0, receive from overall queue.
5	CHR$(*sndr%*)	Sender selection. Ignored if both *l* and *n* = 0. If *l* = 4, *sndr* = 2 x job number to select a particular local sender. If *n* = 8, *sndr* = user link address to select a particular logical link.
6	CHR$(*qual%*)	Normally zero. A special case is described in the discussion below.
7–10	0%	Ignored.
11	CHR$(*chnl%*)	Channel number of buffer to which data is to be delivered (1 to 12). If 0, all data is to be saved or discarded, depending on value of *t* in byte 4.
12	CHR$(0%)	Reserved - must be zero.
13–14	*length%*	Maximum length (in bytes) of data to be returned to the buffer on this Receive call. Ignored if *chnl* = 0.

(continued on next page)

Bytes	Value	Description
15–16	*offset%*	Offset (in bytes) from beginning of buffer where data is to begin. If 0, data is stored starting at the beginning of the buffer. Ignored if *chnl* = 0.
17–20	0%	Reserved - must be zero.
21–26	0%	Ignored.
27–28	*slptime%*	Sleep time, in seconds. If 0, indicates indefinite sleep (see Discussion). Ignored if *s* in byte 4 = 0.
29–40	0%	Reserved - must be zero if passed.

Data Returned:

Formats for the various types of messages that can be returned on a Receive call are described in Sections 5.7.1 - 5.7.9.

Discussion:

rmod

The receive modifier (*rmod*) consists of four flags: $s\% + t\% + l\% + n\%$.

s The sleep flag (*s*) defines what is to be done if there is no appropriate message in the queue for the Receive call to deliver — that is, a message of the type requested with the *l*, *n*, *sndr*, and *qual* arguments. If $s = 1$, the program will "sleep," or suspend execution, until something happens. The *slptime* parameter regulates the time that the program will sleep.

If the *s* flag is set to 0, the program will not sleep if there are no appropriate pending messages. The call terminates immediately with ERR = 5.

t The truncate flag (*t*) is set to 2 to discard any excess data that will not fit in the I/O buffer specified in this call. Information could be left over because (1) the *chnl* parameter is set to 0, or (2) the *length* parameter (or if *length* = 0, the buffer size minus *offset*) is less than the actual amount of data in the message.

A 0 value for *t* indicates that all excess data is to be kept for retrieval on later Receive calls.

l The local sender flag (*l*) is set to 4 to select a message from a local sender. The selection can be further qualified by the *sndr* and *qual* parameters.

If $l = 0$, local selection is not requested. If both *l* and $n = 0$, the first message in the queue is returned.

n The network sender flag (*n*) is set to 8 to select a message from a network sender. The selection can be further qualified by the *sndr* and *qual* parameters.

If $n = 0$, network selection is not requested. If both *l* and $n = 0$, the first message pending in the queue is returned. If $l = 4$, and $n = 8$, local selection prevails.

sndr

The sender selection parameter (*sndr*) is used to select a particular local sender or a particular logical link for this Receive call.

If $l = 4$ (local selection) and *sndr* is nonzero, *sndr* is interpreted as a job number times two. The first message on the queue from that particular local job is retrieved.

If $l = 0$ and $n = 8$ (network selection), a nonzero value for *sndr* is interpreted as a user link address. The first message on the queue from that logical link is delivered.

If $sndr = 0$ when either $l = 4$ or $n = 8$, the *qual* parameter has special meaning. The *sndr* and *qual* parameters are ignored if both l and n are zero.

qual

The sender selection qualifier is normally zero for user applications.

If $l = 4$ (local selection) and $sndr = 0$, then any nonzero value for *qual* is a special-case Receive call, requesting a message from the system (job number 0). This special case is used by the RSTS/E utility program ERRCPY which processes messages from the monitor error logging routines.

The *qual* parameter is ignored if both l and $n = 0$, or if $sndr \neq 0$.

Table 5–4 summarizes the relationships between the selection bits l and n, and the *sndr* and *qual* parameters.

Table 5–4: Sender Selection Summary

rmod				
n	***l***	***sndr***	***qual***	**Result**
0	0	n/a	n/a	The *sndr* and *qual* values are ignored. The Receive call returns the first message in the pending message queue.
n/a	4	0	0	Selects the first Local Data Message in the queue.
		0	nonzero	Selects job 0. This combination is used by the error logging programs to select messages from monitor error logging routines.
		nonzero	n/a	Selects Local Data Messages by job number. *sndr* is interpreted as a job number times 2. Only messages from that job will be delivered on this Receive call.
8	0	0	n/a	Selects the first network message (anything other than a Local Data Message) in the queue.
		nonzero	n/a	Selects network messages from a particular logical link. *sndr* is interpreted as a user link address. Only network messages from the designated logical link will be delivered on this Receive call.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating the buffer to which the data (if any) is to be delivered. Or *chnl* can be 0, indicating that all of this information is to be (1) saved for retrieval in a later Receive call (if *t* in byte 4 = 0), or (2) discarded (*t* = 2).

length

The *length* field is ignored if *chnl* = 0. Otherwise, *length* indicates the maximum number of bytes of information to be delivered to the I/O buffer on this Receive call. If nonzero, this value must be less than or equal to the buffer size minus *offset*.

If *length* = 0, the maximum number of bytes to be delivered to the buffer is equal to the buffer size minus *offset*. (The buffer size is defined by the RECORDSIZE option in the OPEN statement that allocated the buffer.)

If a program has limited buffer space, the *length* value can be used to process a large segment in small pieces. If truncation is not requested (*t* in byte 4 = 0) and the amount of data in the message is greater than the amount indicated by *length*, then only the amount indicated by *length* will be returned. The rest will be saved for retrieval with later Receive calls.

The usual sequence in this case is to issue Receive calls until the number of bytes remaining in the pending message goes to zero. This point can be determined from the control information returned to the target string on a Receive call. Bytes 9–10 returned in the string indicate the number of bytes remaining in the pending message. The number of bytes actually returned to the buffer is given in bytes 13–14 of the returned string.

offset

Received data can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated by this parameter. For example, if *offset* = 20, the data will be delivered beginning at byte 21:

byte 1 20 | 21

offset	user data

whole buffer

The *offset* must be in the range 0 through buffer size minus 1. (The buffer size is defined by the RECORDSIZE option in the OPEN statement that allocated the buffer.)

slptime

The *slptime* parameter is significant only when the *s* flag of *rmod* = 1. In that case, *slptime* defines the length of time, in seconds, that a program is to sleep when no appropriate message is pending in the queue. The program will sleep until:

1. The sleep timer (*slptime*) expires.
2. *Any* new message is placed in the queue (not just an appropriate one).

3. A delimiter is typed on a terminal opened by, or assigned to, this job.
4. Log-ins are disabled. (This could occur if the system is being shut down.)
5. A state change occurs on a pseudo-keyboard assigned to the job. (The job has printed output for the controlling job to read or has entered an input wait state.)
6. The job itself has opened the XM: (DMC11 driver) and a message is received for the job.

In all cases, the program is awakened with an error (ERR = 5) but is not passed a message. To obtain a pending message, the program must execute another Receive call. Since the program can be awakened by any of the conditions listed previously, a check for pending messages can be made by issuing a Receive call without a sleep. A check for terminal input can be made by executing a terminal input operation (GET) using the RECORD 8192% modifier for immediate return. (See the *RSTS/E Programming Manual.*)

Possible Errors:

ERR Value	Error Text and Meaning
5	CAN'T FIND FILE OR ACCOUNT For a Receive call without sleep (*s* in byte 4 = 0), this error indicates that no appropriate messages are pending. For a Receive call with sleep (*s* = 1), this error is returned when the program is awakened from the sleep. The program must execute another Receive call to retrieve any pending messages. (See the previous discussion of the *s* flag for further details.)
9	I/O CHANNEL NOT OPEN A Receive call was issued and the channel specified in byte 11 of the data passed was not open. The program must OPEN the channel before using it for message operations. See Section 5.1.2.
18	ILLEGAL SYS() USAGE No Receiver ID Block exists for the calling program. Before any Receive call can succeed, a Declare Receiver call must be executed.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and *length* fields passed in bytes 13–16 are invalid. The following relationships must be true for a Receive call: 1. The *offset* must be less than the buffer size. 2. The *length* must be less than or equal to the buffer size minus *offset*.

Example:

In the following example, a Receive call is issued for the first message in the queue. The sleep bit is set and the timer is set to three seconds. The truncate bit is 0, as is the channel, so all the information that is available is saved for another Receive call. (The program will examine the third byte of the returned target string to see what type of message is available and GOTO an appropriate section of code depending on its value.)

```
RMOD% = 1%     !SLEEP IF NO MESSAGES
SLPTIME% =3% !SLEEP MAX. OF 3 SECONDS
CHNL% = 0%     !SAVE ALL DATA FOR NEXT RECEIVE
!
!EXAMPLE OF RECEIVE CALL
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(2%) + CHR$(RMOD%)
        + STRING$(6%,0%)
        + CHR$(CHNL%) + STRING$(15%,0%)
        + CHR$(SLPTIME%) + CHR$(SWAP%(SLPTIME%)))
```

5.7.1 Format of Received Local Data Message

The format of the information returned to the target string for a Local Data Message is described below. Up to 512 bytes of user data can be delivered to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1-2	—	Not meaningful - should be ignored.
3	CHR$(-1%)	Local Data Message subfunction code.
4	CHR$(*jobx2%*)	Two times the job number of the local sender.
5-6	*ppn%*	Project-programmer number of the local sender. Byte 5 = programmer number, byte 6 = project number.
7-8	—	Not meaningful - should be ignored.
9-10	*remainder%*	Number of bytes of user data not delivered to I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set.
11-12	—	Not meaningful - should be ignored.
13-14	*length%*	Length (in bytes) of information transferred to the I/O buffer on this Receive call.
15-20	—	Not meaningful - should be ignored.
21-40	*param$*	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 20 bytes.

5.7.2 Format of Received Connect Initiate Message

The format of the information returned to the target string is described below. A 120-byte Connect Data Block and up to 16 bytes of user data are also returned to the I/O buffer specified in the Receive call. The format of a received Connect Data Block is shown in Figure 5–2 and described in detail in Table 5–5.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	CHR$(-2%)	Connect Initiate Message subfunction code.
4	—	Not meaningful - should be ignored.
5–6	*lla%*	Local link address. Identifies the link for the local program's subsequent Connect Confirm or Connect Reject Message.
7–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	Length (in bytes) of data transferred to the I/O buffer on this Receive call.
15–22	—	Not meaningful - should be ignored.
23	CHR$(*rlmod%*)	Remote link modifier. Indicates the flow control in effect for the remote program. The two low-order two bits should be masked and interpretted as follows: *flow%* = *rlmod%* AND 3% If *flow* = 0, no flow control = 1, segment flow control = 2, message flow control
24	—	Not meaningful - should be ignored.
25–26	*rmax%*	Receive maximum, in bytes. The maximum amount of user data that will be transmitted to the local program in one Network Data Message over this link. The *rmax* value is supplied by the local NSP and is calculated from the size of the receive buffers it allocates. The maximum size of the receive buffers is set for NSP by Network Management. The local program can further limit the amount of user data to be received by specifying a smaller *rmax* in the Connect Confirm Message that accepts this request for a connection. (See Section 5.6.3.)
27–28	*tmax%*	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote NSP. The local NSP enforces this limit for all Network Data Messages sent by the local program on this link.
29–40	—	Not meaningful - should be ignored.

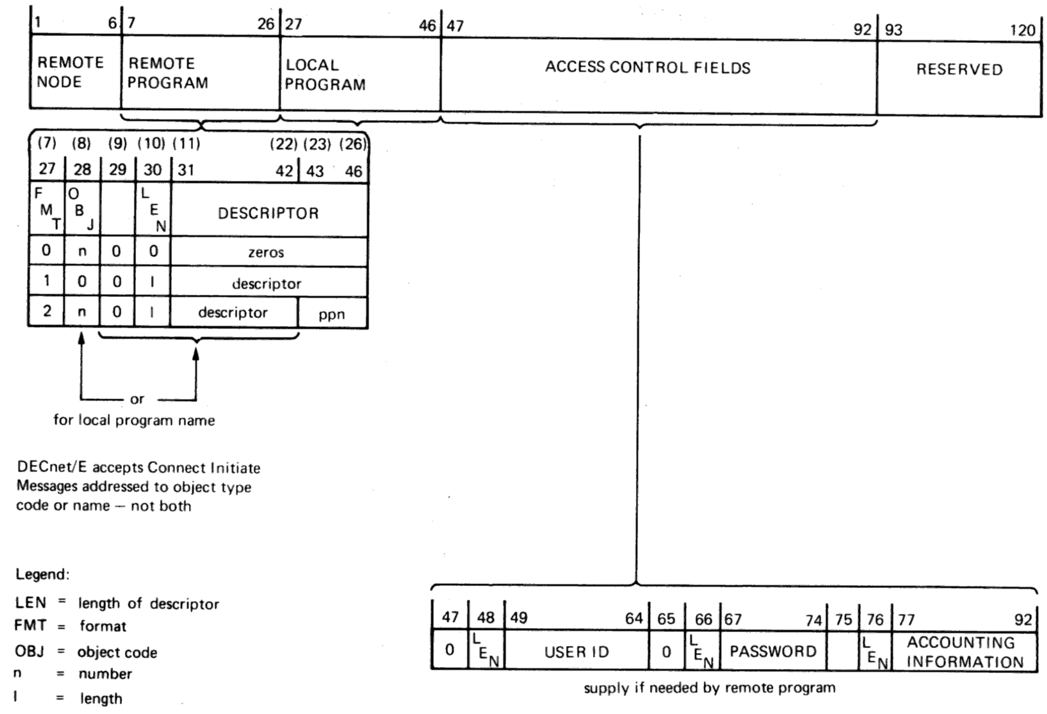


Figure 5–2: Format of Received Connect Data Block

Table 5–5: Format of Received Connect Data Block

Bytes	Content
1–6	**Remote Node**. 1 to 6 uppercase, alphanumeric characters giving the name of the remote node from which the Connect Initiate Message was received. The name will contain at least one alphabetic character. Node names shorter than six characters are left-justified and padded to six characters with spaces.
7–26	**Remote Program**. Identifies the remote program that sent the Connect Initiate Message as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of three formats. **Format 0** The remote program is identified by object type code alone. The fields are: **Bytes** **Content** 7 Zero, to indicate a format 0. 8 Object type code of the remote program. 9–26 Zero (not used). **Format 1** The remote program is identified by a descriptor. The subfields are:

(continued on next page)

Bytes	Content
7	One, to indicate a format 1.
8	Normally zero. This byte will usually be zero unless the remote DECnet system identifies the remote program with both name and object type code on outgoing connect requests. If nonzero, this byte is interpreted as the object type code of the remote program that sent the Connect Initiate Message.
9	Reserved - must be zero.
10	Length (in bytes) of descriptor, 0 to 16. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-26).
11-26	Remote program descriptor (left-justified). The descriptor used to register the remote program for network operations.

Format 2

This format identifies the remote program by name or object type code and by the group and user codes under which the remote program is running on the remote system. (The group and user codes refer to what is called the project-programmer number on some DIGITAL systems, including RSTS/E. Other DIGITAL systems may refer to these codes as the user identification code, or UIC.) DECnet/E systems use this format to identify the sender of a Connect Initiate Message.

The subfields are defined as follows:

Bytes	Content
7	Two, to indicate a format 2.
8	Object type code of the remote program, if applicable. If zero, the descriptor alone identifies the remote program.
9	Reserved - must be zero.
10	Length of descriptor, 0 to 12. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
11-22	Remote program descriptor, if this is used instead of object type code. The descriptor will be left-justified ASCII characters.
23-24	Remote group code. Binary value giving the group number under which the remote program is running. If the remote system is a DECnet/E system, byte 23 is the project number in a RSTS/E project-programmer number.
25-26	Remote user code. Binary value giving the user number under which the remote program is running. If the remote system is a DECnet/E system, byte 25 is the programmer number in a RSTS/E project-programmer number.

27-46 **Local Program Name**. This field indicates how the remote program addressed the local program in the Connect Initiate Message. DECnet/E allows the local program to be addressed by object type code or by name, but not both. Again, three formats can be used. The subfields are as follows:

Bytes	Content
27	Format type. Equal to 0, 1, or 2, depending on the format used by the remote program to address the local program.
28	Object type code, if used by the remote program to address the local program, (formats 0 or 2).
29	Always zero.

(continued on next page)

30	Length of name, in bytes. Will be 0 for format 0, 1 to 6 for format 1, 0 to 6 for format 2. (DECnet/E program names are limited to a maximum of 6 characters by the Declare Receiver call.)
31–36	Local program name, if used by the remote program to address the local program. (formats 1 or 2).
37–42	Always zero.
43–44	Local group code. Binary value giving the local program's project value, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by NSP to identify the local program but passed on here.
45–46	Local user code. Binary value giving the local program's programmer value, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by NSP to identify the local program but passed on here.

47–92 **Access Control Fields**. These bytes can be used to define the remote program's access rights to the local program's services. The idea is analogous to logging into the local system. The local program could use this information to log into the RSTS/E system, grant classes of privileges, or whatever. The most rigorous checking done in a normal log-in on DIGITAL systems involves user identification, password, and account number, so three fields are defined here for those items. However, DECnet does not specifically require use of these fields at all, nor does it require that the fields be used as described. A DECnet/E system simply passes this information onto the receiving program.

Bytes	Content
47	Not meaningful – should be ignored.
48	Length of user identification information specified in the next field, 0 to 16 bytes.
49–64	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the project-programmer number used at log-in.
65	Not meaningful – should be ignored.
66	Length of password information specified in the next field, 0 to 8 bytes.
67–74	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the project-programmer number at log-in.
75	Not meaningful – should be ignored.
76	Length of the accounting information specified in the following field, 0 to 16 bytes.
77–92	Accounting Information. Many systems require a billing account number in addition to user identification and password. The accounting field is provided for this purpose.

93–120 **Not meaningful** – should be ignored.

5.7.3 Format of Received Connect Confirm Message

A received Connect Confirm Message is a remote program's acceptance of a Connect Initiate Message sent by the receiving program. The link is identified by the user link address specified by the local program in the Connect Initiate Message. The format of the information returned to the target string is described below. Up to 16 bytes of user data can also be returned to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	CHR$(-3%)	Connect Confirm Message subfunction code.
4	CHR$(*ula%*)	User link address.
5–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive, depending on how the *t* flag was set in the data passed.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	Length (in bytes) of data that was returned to the I/O buffer on this Receive.
15–22	—	Not meaningful - should be ignored.
23	CHR$(*rlmod%*)	Remote link modifier. Indicates the flow control in effect for the remote program. The two low-order bits should be masked and interpretted as follows: *flow%* = *rlmod%* AND 3% If *flow* = 0, no flow control = 1, segment flow control = 2, message flow control
24	—	Not meaningful - should be ignored.
25–26	*rmax%*	Receive maximum, in bytes. The maximum amount of user data that will be transmitted to the local program in one Network Data Message over this link. This will be (1) equal to the *rmax* value specified in the Connect Initiate Message for this link, or (2) a lesser value supplied by the local NSP, calculated from the size of the buffers that it allocates for receives.
27–28	*tmax%*	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote NSP. The local NSP enforces this limit for all Network Data Messages sent by the local program on this link.
29–40	—	Not meaningful - should be ignored.

5.7.4 Format of Received Connect Reject Message

A received Connect Reject Message is a rejection of the receiving program's request for a logical link. It can be from the remote program or from the local or remote NSP. The link is identified by the user link address previously established in the local program's Connect Initiate Message. The format of the information returned to the target string is described below. Rejections from NSP contain a reason code explaining the reason for the rejection and do not contain user data. Rejections from the remote program can contain up to 16 bytes of user data delivered to the I/O buffer.

Bytes	Value	Description
1-2	—	Not meaningful - should be ignored.
3	CHR$(-4%)	Connect Reject Message subfunction code.
4	CHR$(*ula%*)	User link address, previously defined in the local program's Connect Initiate Message.
5–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the data passed.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	Length (in bytes) of data returned to the I/O buffer on this Receive call.
15–22	—	Not meaningful - should be ignored.
23–24	*reason%*	Identifies the reason for the rejection (see Appendix B).
25–40	—	Not meaningful - should be ignored.

5.7.5 Format of Received Network Data Message

A received Network Data Message contains user data from a remote program. The format of the information returned to the target string is described below. The user data accompanying the Network Data Message is delivered to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	CHR$(-5%)	Network Data Message subfunction code.
4	CHR$(*ula%*)	User link address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the data passed.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	Length (in bytes) of data that was returned to the I/O buffer on this Receive call.
15–20	—	Not meaningful - should be ignored.
21	CHR$(*dmflgs%*)	Data message flags. The two low-order bits indicate whether this is the beginning, middle, end, or sole segment of a logical message. These bits should be masked and interpretted as follows: *bom%* = *dmflgs%* AND 1% *eom%* = *dmflgs%* AND 2% Then *eom* *bom* 0 0 Middle segment of message 0 1 First segment of message 2 0 Last segment of message 2 1 Sole segment of message DECnet/E uses the *eom* bit to determine the end of a logical message if the local program requested message flow control when the logical link was established. It does not use the *bom* bit but passes it on to the receiving program.
22–40	—	Not meaningful - should be ignored.

5.7.6 Format of Received Interrupt Message

The local NSP places Interrupt Messages from a remote program at the head of the pending message queue, behind the first message and behind any other pending Interrupt Messages queued for the program. The format of the data returned to the target string is described below. Up to 16 bytes of user data can be delivered to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	CHR$(-6%)	Network Interrupt Message subfunction code.
4	CHR$(*ula%*)	User link address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for later Receive calls, depending on how the *t* flag was set in the data passed.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	Length (in bytes) of data returned to the I/O buffer on this Receive call.
15–40	—	Not meaningful - should be ignored.

5.7.7 Format of Received Link Service Message

A Link Service Message provides status information for a particular logical link and is only returned on a Receive call when the pending message queue has emptied after a condition has cleared that previously inhibited the local program from sending (see Section 4.2). Status information with the same format as a Link Service Message can also be returned to the target string on successful completion of a Send Link Service, Send Network Data, or Send Interrupt call. The format of the information returned to the target string is described below. No data is delivered to the I/O buffer with a Link Service Message.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	CHR$(-7%)	Link Service Message subfunction code.
4	CHR$(*ula%*)	User link address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5–8	—	Not meaningful - should be ignored.
9–10	0%	Always zero for Link Service. (These bytes are the *remainder* field in other returned messages.)
11–12	—	Not meaningful - should be ignored.
13–14	0%	Always zero for Link Service. (These bytes are the *length* field in other returned messages.)
15–20	—	Not meaningful - should be ignored.
21	CHR$(*lstat%*)	Local status flags. These flags should be masked and interpreted as follows: *data%* = *lstat%* AND 1% *intls%* = *lstat%* AND 2% *lclbp%* = *lstat%* AND 192% (2-bit field) If *data* = 1, outgoing Network Data Messages are inhibited. (A Link Service Message will be delivered when the condition clears.) If *intls* = 2, outgoing Interrupt and Link Service Messages are inhibited. (A Link Service Message will be delivered when the condition clears.) The *lclbp* value indicates the local NSP's backpressure status: 0 Incoming Data Message flow is on. 64 Incoming Data Message flow is on but the local NSP will turn this link off if another Data Message is received before the local program's receive queue is emptied.

Bytes	Value	Description
		128 Incoming Data Message flow has been turned off by the local NSP. The local program's receive queue must be emptied before the link will be turned on. 192 Incoming Data Message flow has been turned off by the local NSP but flow is scheduled to turn on as soon as system buffers become available.
22	CHR$(*rstat%*)	Remote status flags. Only one bit is currently defined for this field. This bit should be masked and interpretted as follows: *rembp%* = *rstat%* AND 128% If *rembp* = 128, the link has been turned off by the remote system due to a backpressure condition. No Data Messages can be sent until the remote system turns the link back on.
23	CHR$(*ldr%*)	Local data request count. The count of segments or logical messages currently requested by the local program, reflecting the actual number of segments or logical messages requested but not yet received.
24	CHR$(*lir%*)	Local interrupt request count. The count of Interrupt Messages currently requested by the local program, reflecting the actual number of interrupts requested but not yet received. This count will never exceed one.
25	CHR$(*rdr%*)	Remote data request count. The count of segments or logical messages currently requested by the remote program, reflecting the actual count of segments or messages requested but not yet sent.
26	CHR$(*rir%*)	Remote interrupt request count. The count of Interrupt Messages currently requested by the remote program, reflecting the actual count of Interrupt Messages requested but not yet sent.
27	CHR$(*dtm%*)	Data transmit queue maximum. The maximum number of data messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when NSP is enabled. It applies individually to all active logical links.
28	CHR$(*dtc%*)	Data transmit queue count. The number of data messages currently in the data transmit queue for this logical link. It is a count of data messages waiting for acknowledgment from the remote NSP.
29	CHR$(*ltm%*)	Interrupt/Link Service transmit queue maximum. The maximum number of Interrupt or Link Service Messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when NSP is enabled. It applies individually to all active logical links.
30	CHR$(*ltc%*)	Interrupt/Link Service transmit queue count. The number of Interrupt and Link Service Messages currently in the Interrupt/Link Service transmit queue for this logical link. It is a count of Interrupt and Link Service Messages waiting for acknowledgment from the remote NSP.

(continued on next page)

Bytes	Value	Description
31	CHR$(*mmax%*)	Message maximum. The maximum number of incoming messages that will be queued for the local program. This maximum is set by the program in the Declare Receiver call.
32	CHR$(*mcnt%*)	Message count. The total number of messages currently in the pending message queue for the local program.
33	CHR$(*mula%*)	Message count for this logical link. The number of messages currently in the pending message queue for this logical link, as identified by the *ula* (byte 4).
34–40	—	Not meaningful - should be ignored.

5.7.8 Format of Received Disconnect Message

A received Disconnect Message indicates that an established logical link with a remote program has been disconnected by the remote program. All other messages sent by the remote program over this logical link have been received by the local program. The format of the data returned to the target string is described below. Up to 16 bytes of user message data can be returned to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	CHR$(-8%)	Disconnect Message subfunction code.
4	CHR$(*ula%*)	User link address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	Number of bytes of user data not returned to I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the data passed.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	Length (in bytes) of data delivered to the I/O buffer on this Receive call.
15–40	—	Not meaningful - should be ignored.

5.7.9 Format of Received Link Abort Message

A received Link Abort Message indicates that a logical link has been terminated by the remote program or by the local or remote NSP. The format of the data returned to the target string is described below. Up to 16 bytes of user data can be returned to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1-2	—	Not meaningful - should be ignored.
3	CHR$(-9%)	Link Abort Message subfunction code.
4	CHR$(*ula%*)	User link address (if any), previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-6	*lla%*	Local link address, established by the local NSP for this link. It identifies the link if the remote program has sent a Connect Initiate Message but the local program has not yet responded with a Connect Confirm or Connect Reject Message. No *ula* exists to identify the link being aborted under these conditions.
7-8	—	Not meaningful - should be ignored.
9-10	*remainder%*	Number of bytes of user data not transferred to I/O buffer. This data has been either discarded or saved for later Receive calls, depending on how the *t* flag was set in the data passed.
11-12	—	Not meaningful - should be ignored.
13-14	*length%*	Length (in bytes) of data transferred to the I/O buffer on this Receive call.
15-22	—	Not meaningful - should be ignored.
23-24	*reason%*	Abort Reason (interpreted as a 16-bit integer). If this value is nonzero, the link was aborted by either the local or remote NSP and the reason is specified in these two bytes. The reasons that apply for a link abort are listed in Appendix B. If *reason* is zero, the link was aborted at the request of the remote program.
25-40	—	Not meaningful - should be ignored.

Part IV
Network Programming in BASIC–PLUS–2

Chapter 6
Network Programming In BASIC–PLUS–2

This chapter presents the specific calls, with detailed formats, used to access the DECnet/E message services in the BASIC–PLUS–2 programming language.

6.1 Programming Background

To use the RSTS/E message services in BASIC–PLUS–2, you code subroutine calls with the CALL BY REF statement. The calls reference external MACRO subroutines stored in the system library that must be linked with your program. Thus, the process involves coding, compiling, and linking. These aspects of using DECnet/E in BASIC–PLUS–2 are described briefly in the following subsections. Specific call formats are given in Sections 6.2 – 6.7.

6.1.1 Coding

The general form of the RSTS/E message calls in BASIC–PLUS–2 is:

CALL *name* BY REF(*arg1*, *arg2*, ...)

where

name — is the name of one of the 13 subroutines available for message services. Each subroutine processes a call at execution time by:

1. Checking for the proper number of arguments in the call.
2. Translating the call and arguments into the form recognizable to the RSTS/E monitor.
3. Passing control to the RSTS/E monitor which in turn invokes the NSP software. (In DECnet/E systems, NSP is part of the RSTS/E monitor.)

NSP processes the request and returns control to the user program. If an error occurs, NSP returns an error value to an integer variable defined as an argument in each call.

arg... is the argument list for the particular call. Values passed can be any valid BASIC-PLUS-2 expression of the appropriate type. For example, if the description of an argument calls for an integer value, you can supply any valid BASIC-PLUS-2 integer constant, variable, or expression.

NOTE

CALLs, as opposed to FUNCTIONs, do not do implicit type conversions. That is, if an integer value is required and you specify a floating point value, the floating point value will be passed to the message subroutine. Unexpected fatal errors can result.

Most calls contain arguments to define buffer areas for sending and receiving data. A discussion follows on how buffers can be set up for send and receive calls.

Setting Up Buffers

You specify either two or three arguments in the DECnet/E message calls to define a buffer: a starting address, a length value defining the number of bytes in the buffer, and in some cases, an offset in bytes from the beginning of the buffer. In all cases, the sends and receives use a contiguous area in computer memory to send from or receive to. Thus, for most applications, it is better to preallocate space for send and receive buffers rather than to use dynamic strings or arrays. The following discussion addresses both methods.

Preallocating Space

Use the DIM, COM, or MAP statements to allocate a contiguous block of memory at compile time for send and receive buffers. The MAP statement is particularly useful. It allocates space and allows the definition of fields within the buffer having different data types. You can, for example, define a map with a string field, followed by numeric values, followed by another string. The same area can also be remapped with different field definitions. For example:

```
MAP (RETBUF) A$=40%
MAP (RETBUF) FILL%,CODEULA%,LLA%,FILL%,REMAINDER%, &
             FILL%,LENGTH%,FILL$=6%,PARAM$=20%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,LLA%,FILL%,REMAINDER%, &
             FILL%,LENGTH%,FILL$=6%,PARAM$=20%
```

The first MAP statement allocates a 40-byte area. The MAP name is RETBUF and A$ can serve as the starting address argument for a message call.

The next two MAP statements illustrate two ways of getting at integer byte values in BASIC-PLUS-2. The second MAP statement shows an integer field CODEULA%, a one-word field that is to be referenced as a one-byte code

followed by a one-byte user link address. Once data is in the buffer, you can reference the code byte with a statement of the form:

```
CODE%=SWAP(CODEULA%) AND 255%
```

And you can reference the *ula* byte with a statement of the form:

```
ULA%=CODEULA% AND 255%
```

The third MAP statement shows the same area in the buffer as two one-byte string variables, CODE$ and ULA$. You can then refer to bytes three and four as integer values with statements of the form:

```
CODE%=ASCII(CODE$)
ULA%=ASCII(ULA$)
```

Using Dynamic Strings and Arrays

A dynamic string or array is any string or array for which space has not been preallocated with a DIM, COM, or MAP statement. BASIC–PLUS–2 allocates space for dynamic strings and arrays at execution time, not at compile time. Except for dynamic string arrays, you can use dynamic values for send requests, as shown in the following section, "Buffer Arguments." The elements of dynamic string arrays are not stored contiguously in memory. Hence, a send request referring to an entire dynamic string array could include data that is not part of the array itself.

For receive requests, dynamic strings and arrays should not be used at all. The amount of data stored in a buffer by a receive request is limited only by the length argument in the Receive call. It is not limited by the length of the dynamic string or array. Hence, if the data returned by the Receive call is longer than the space currently allocated for the dynamic string or array, the data stored by the receive will destroy some quantity of data in locations that are uncontrolled by the user program.

Buffer Arguments

Starting Address: For buffers from which data is to be sent, you can specify any valid BASIC–PLUS–2 constant or variable for this argument, except a variable naming an entire dynamic string array.

If you wish to send pure numeric data, for example, you could specify this argument as an array name in the form A(), B%(,), and so forth.

NOTE

BASIC–PLUS–2 stores two-dimensional array values in contiguous locations by row — (0,0) first, (0,1) second, (0,2) third, and so forth. This can be significant if the data is received by a remote FORTRAN program that assumes two-dimensional arrays are stored by column.

It can also be practical to specify a string constant as the starting address argument for certain types of sends. For example, suppose you wish to notify the remote program of a reason for aborting a logical link. You can code the call with the string constant "ENDFILE" as the starting address argument. You must ensure that the length argument is correct in these cases, however.

For buffers that are to receive data, the starting address argument should name some variable for which space has been preallocated with DIM, COM, or MAP, as discussed previously.

Offset: This argument can be any valid integer value. It defines an offset (in bytes) from the starting address where the data to be sent begins. This is useful if you are sending segments from a buffer containing an entire logical message.

Length: This argument can be any valid integer value. It defines the total number of bytes to be sent or the maximum number of bytes to be received. Remember that each character of a string is 1 byte, an integer value is 2 bytes, a single-precision floating point value is 4 bytes, and a double-precision floating point value is 8 bytes.

6.1.2 Compiling and Linking

You can compile BASIC–PLUS–2 programs that use the message services to run with either the BASIC2 or BP2COM Run-Time System by using the COMPILE command. You must use the /OBJECT switch since external routines must be linked to the object module produced by compilation to form an executable file. You would then use the BUILD command to produce a command file and overlay descriptor file for the Task Builder. For example, for a source file named NETCOM.B2S stored on the public disk structure, the compilation process would be:

```
OLD NETCOM               (default extension = .B2S)
COMPILE/OBJECT NETCOM    (default extension = .OBJ)
BUILD NETCOM             (default extensions = .CMD and .ODL)
```

In the above sequence, the OLD command reads the source file NETCOM.B2S from disk into memory where it can be compiled. The COMPILE command with the object switch produces an object module file NETCOM.OBJ. (Object module simply means the object code addresses are relocatable. Several object modules can then be linked together with the Task Builder to produce an executable program.)

The BUILD command produces two files: a command file (NETCOM.CMD) and an overlay descriptor file (NETCOM.ODL).

You can then link the program with

```
TKB @NETCOM
```

The Task Builder (TKB) will automatically resolve the references to the message subroutines from the system library (LB:SYSLIB.OLB), producing the executable file NETCOM.TSK.

6.2 Declare Receiver (MDCL) — Privileged (Local and DECnet)

The MDCL call must be executed before a program can receive any messages or send any network messages. (Local messages can be sent without first executing this call.) The call defines an identifying object type or name, or both, and any restrictions on incoming messages. The monitor associates this information with the RSTS/E job number, setting up a Receiver ID Block for the job, as described in Section 3.2.3. Only one MDCL call can be in effect at a time for a particular job. It remains in effect until a MREM (Remove Receiver) call is executed for the job.

BASIC–PLUS–2 Call Format:

CALL MDCL BY REF(*errval%,name$,objtyp%,access%,*
bmax%,mmax%,lmax%)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MDCL call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

name

The logical name is a string consisting of one to six characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underline). If less than six characters are used, the name must be left-justified in the string and the remaining bytes must be padded with spaces. (A store into a 6-byte string defined in COMMON or MAP will do this automatically.) Leading or embedded spaces are invalid. If no name is to be specified (that is, the logical name is null), the string should contain 6 spaces.

Local programs can address the calling program by the logical name. Network programs can use either the logical name or the object type code, if one is declared.

For local or network programs to access the calling program by logical name, the logical name must be nonnull and unique to the node. That is, only one program at the node can declare its identity with that logical name at any given time. NSP uses the logical name to identify the calling program for queuing messages from local or network programs. If the logical name is null, only network access by object type is allowed. (See the following discussion of *name*, *objtyp*, and *access*.)

objtyp

This integer value specifies the object type code, another form of network addressing. As discussed in Section 3.2.1, the object type code defines some service the program performs. If the program issuing the MDCL call is addressed by object code alone (*name* is null and *access* indicates network access only), multiple copies can execute at the same time in the RSTS/E environ-

ment. Unlike the logical name, the object type code need not be unique within the node — multiple copies of a program can declare their identity with the same type code at the same time. If *name* is null, *objtyp* cannot be zero. (Network addressing is discussed in detail in Section 4.1.)

Acceptable values for the object type code range from 0 through 255. The value 0 indicates that other programs in the network will never address the calling program by object type code. The range from 1 to 127 is reserved for DECnet use (see Appendix A). The range from 128 to 255 is available for definition and use by a network installation.

access

This integer value consists of the sum of five bit values ($l\% + p\% + n\% + o\% + s\%$) that are used to determine access to the declaring program and to modify certain aspects of the message operations.

Three of these bit values ($l\%$, $p\%$ and $n\%$) limit the types of senders that can queue messages for the calling program. They do not, however, limit the messages that the program can send.

If $l = 0$, messages from local senders will not be queued for the calling program. (Local senders who use the network functions are considered network senders in this context.) If $l = 1$, messages from local senders will be queued.

If $p = 0$, incoming messages from both local privileged and nonprivileged senders will be queued for the calling program. If $p = 2$, incoming messages from local senders will be queued only if the sender is privileged. (This bit is ignored if $l = 0$, that is, if no local senders are allowed.)

If $n = 0$, incoming requests for logical links are rejected by NSP. If $n = 4$, incoming Connect Initiate Messages are passed on to the receiver, subject to the declared message maximum (*mmax*) and the declared link maximum (*lmax*). In this case, the program must provide its own protection from unauthorized access, if necessary.

Table 6–1 summarizes access code values and the access permitted for each.

Table 6–1: Types of Receiver Access

Access Code (n%+p%+l%)	Network Logical Links	Local Senders Privileged	Local Senders Allowed
0	no	no	no
1	no	no	yes
2	no	no	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	no	no
7	yes	yes	yes

The bit value *o* is used to regulate single-link programs, modifying the function of the program's declared link maximum (*lmax*). If $o = 0$, incoming requests for logical links do not affect the program's link maximum. If $o = 8$, however, after queuing an incoming Connect Initiate Message to the receiver, NSP sets the program's link maximum to zero, inhibiting all further incoming connect requests. This, in effect, modifies the meaning of the link maximum from "one link at a time" to "one link per program execution."

By setting this "one-shot" bit value to 8, a program prevents the possibility that NSP will queue a Connect Initiate Message for it after its one logical link is disconnected and before it issues a Remove Receiver call.

For example, the DECnet/E FAL utility sets the one-shot bit to 8 and declares a link maximum of 1. Once FAL is started as the result of an incoming request from a remote NFT utility, NSP queues the Connect Initiate Message and zeros the link maximum. Thus, there is no possibility that an incoming connect request can be queued for that copy between the time that NFT breaks the link and the time that FAL issues its Remove Receiver call. This ensures that each incoming connect request starts a new copy of FAL.

Bit value *s* is used to modify the function of the RSTS/E conditional Sleep monitor call. If $s = 0$, any unreceived message queued for the program will block the execution of a conditional Sleep call. This is the normal RSTS/E function. If $s = 16$, however, the RSTS/E monitor will not check the program's message queue when determining whether or not it should in fact suspend program execution. Several other conditions (such as a delimiter received on an open terminal) can block the Sleep call but a pending message will not.

logical name, objtyp, and *access*

NSP checks the logical *name*, *objtyp*, and *access* arguments to ensure that they are consistent. For example, a null logical name and an access allowing local senders are inconsistent since local senders can address the program by logical name. NSP checks these arguments as shown in Table 6–2. If the arguments are invalid, NSP returns an error code of 18 in the *errval* argument. If the arguments are valid, NSP allocates a small buffer to hold the information passed. If no buffers are available, the call fails with *errval* = 4. A retry may succeed.

bmax

Until a Receive call (MRCV) is executed, a pending message occupies system buffer space. One 16-word buffer from the monitor's buffer pool is used for user- or DECnet-defined parameters and other system-specific information. Additional buffer space to hold message data is usually allocated from the extended buffer pool. If an extended pool does not exist or no space is available there, the monitor's buffer pool is used.

Because the monitor pool is a critical system resource, the program must set a limit on the amount of space to be used on its behalf. The integer argument *bmax* sets a limit (1 to 32767 bytes) on the total monitor pool space to be used for the user data portion of messages.

When the number of bytes of monitor pool space used to store message data exceeds the calling program's declared buffer maximum, NSP will no longer queue Local Data Messages for the program. An error will be returned to any local program that tries to send a Local Data Message to this one. Network messages will be queued regardless of the declared buffer maximum but any monitor pool space used will be counted against the maximum.

A zero or negative value indicates that the monitor's buffer pool is not to be used for the user data portion of pending Local Data Messages.

Table 6–2: System Validation of Name, Object, and Access Parameters

Access Code (n%+p%+l%)	Senders Permitted	Object Type	Logical Name
0,2	None	Ignored	Ignored.
1,3	Local only	Ignored	Must be nonnull and unique.
4,6	Network only	Zero	Must be nonnull and unique.
		Nonzero	If the logical name is null (at least one leading binary zero byte), the only access permitted is by object type code. Multiple copies of the program can coexist. If the logical name is nonnull, it must be unique at the local node. Network senders can refer to the calling program by name or by object type code. Only one copy of the program using this logical name can execute at any given time.
5,7	Network/Local	Any value	Must be nonnull and unique so that local senders can refer to the calling program by logical name. Network senders can use the logical name or object type code (if *objtyp* $\neq$ 0). Only one copy of the program using this logical name can execute at any given time.

mmax

The message maximum (*mmax*), an integer value, limits the number of messages that will be queued for the calling program. The value can range from 0 to 255.

NSP keeps a count of the total number of messages queued for each declared receiver. When one of the these counters equals or exceeds the message maximum set by the receiver:

1. Incoming Local Data Messages are not queued for the receiving program. An error is returned to the local sender.
2. Incoming Connect Initiate Messages cause NSP to determine if another copy of the program can be started automatically (see Section 4.1.2). If not, NSP rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

3. An incoming Network Data Message causes NSP to inhibit incoming data messages on that particular logical link and to negatively acknowledge (NAK) the message. This forces the remote system to retransmit when flow is reenabled. (See Section 4.2.2 for further details on backpressure flow control.)

lmax

The link maximum (*lmax*), an integer value, limits the number of incoming requests for logical links for the calling program. If the number of links currently active is greater than or equal to *lmax* (0 to 63) when a remote Connect Initiate Message is received for the program, the local NSP determines if another copy of the program can be started automatically (see Section 4.1.2). If not, NSP rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

By declaring a small link maximum, a program can avoid the overhead of responding to remote Connect Initiate Messages when it is known beforehand that the program can only handle a limited number of logical links.

A zero value for *lmax* has the same effect as setting $n = 0$ in the access argument. That is, NSP rejects all incoming requests for logical links to the program. The *lmax* argument does not limit the number of logical links initiated by the program. That is, it does not stop the program from sending Connect Initiate Messages.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
3	ACCOUNT OR DEVICE IN USE The calling job already exists in the system's list of declared receivers. This error can indicate a logic error in the program or that a previous program running under the same job number failed to remove itself from the receiver list before terminating. In the latter case, a Remove Receiver call should be issued, followed by another Declare Receiver call. (It is common practice to code a Remove Receiver call immediately before the Declare Receiver call.)
4	NO ROOM FOR USER ON DEVICE There were no small buffers available to hold the arguments passed with the Declare Receiver call. Since the system's use of small buffers is dynamic, a retry may succeed.
10	PROTECTION VIOLATION The calling program must be privileged at the time the Declare Receiver call is executed.
16	NAME OR ACCOUNT NOW EXISTS The logical name passed is being used by another job.
18	ILLEGAL SYS() USAGE The parameters passed are inconsistent. See *name*, *object*, and *access* discussion.

Example:

The following example shows a receiver declaration allowing both network and local access. Both a logical name (BUS) and an object type code (142) are declared. Note that 6 bytes for the logical name were preallocated with COM. Thus, when the three characters "BUS" are stored in LNAME$, they are stored left-justified with space fill.

The buffer maximum is set to 512 and the message maximum is set to 5. No more than 512 bytes of user message data from incoming Local Data Messages will be stored in monitor pool space. And, if more than 5 messages accumulate for the program, an incoming request for a connection will cause NSP to check to see if another copy of the program can be automatically started. If not, NSP will reject the link and any further incoming data messages will cause NSP to inhibit incoming message flow on all the program's logical links because of a backpressure condition. The link maximum is set to 2. When the program has established two logical links, an incoming Connect Initiate Message will cause NSP to check to see if another copy of the program can be started. If not, NSP will reject the link.

```
1000 COM LNAME$ = 6% &
     \ERRVAL% = 0% &
     \LNAME$ = "BUS" &
     \OBJTYP% = 142% &
     \ACKSES% = 5% &
     \BMAX% = 512% &
     \MMAX% = 5% &
     \LMAX% = 2% &
        .
        .
        .
2000 CALL MDCL BY REF(ERRVAL%,LNAME$,OBJTYP%,ACKSES%, &
          BMAX%,MMAX%,LMAX%) &
     \IF ERRVAL%<>0% THEN GOTO 12000
        .
        .
```

6.3 Remove Receiver (MREM) (Local and DECnet)

The MREM call removes the specified job from the system's list of declared receivers. All pending messages are discarded and any active logical links are aborted. This call should be executed at the completion of message processing. This prevents unwanted messages from accumulating in the queue of pending messages.

Privileged programs can remove other RSTS/E jobs with this call although normally, only system utility programs would do this.

BASIC–PLUS–2 Call Format:

CALL MREM BY REF(*erral%,jobx2%*)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MREM call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

jobx2

A zero value for *jobx2* removes the calling job. Privileged programs can remove another local RSTS/E job by specifying job number times two for this integer argument.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
10	PROTECTION VIOLATION The caller is nonprivileged and has attempted to remove another job. That is, *jobx2* is nonzero.
18	ILLEGAL SYS() USAGE The *jobx2* argument was odd. *Jobx2* must be zero to remove the calling job, or job number times two to remove another job.

Example:

The following example shows a typical Remove Receiver request removing the calling job from the system's list of declared receivers.

```
\ERRVAL% = 0% &
\CALL MREM BY REF(ERRVAL%,0%)
```

6.4 Get Local Node Parameters (NTLN) (DECnet)

The NTLN call return information to the calling program concerning the local node's network parameters. Data returned includes the node's name, address and alias (if any), and the default user's account to be used when an incoming Connect Initiate Message does not specify accounting information. (See the *DECnet/E System Manager's Guide* for a discussion of alias and Section 4.1.4 of this manual for a discussion of the default user's account.)

BASIC–PLUS–2 Call Format:

CALL NTLN BY REF(*erval%,buffer$,buflen%,bufoff%*)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTLN call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

buffer

This argument defines the starting address of the buffer to receive the local node information. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by *buffer*.

When the NTLN call completes successfully, data is returned to the buffer in the following format:

Bytes	Contents
0–5	Local node name, consiting of 1 to 6 uppercase, alphanumeric ASCII characters. The name contains at least one alphabetic character. A name shorter than 6 characters is left-justified and padded to 6 characters with spaces.
6–7	Local node address, integer value ranging from 1 to 255.
8–13	Local node alias, consisting of 1 to 6 uppercase, alphanumeric ASCII characters. The alias contains at least one alphabetic character. An alias shorter than 6 characters is left-justified and padded to 6 characters with spaces. If no alias is defined, a field of 6 spaces is returned.
14–15	Local default project-programmer number. Byte 14 = program number; byte 15 = project number. If no default account is defined, both bytes are zero.

buflen

This integer argument defines the number of bytes to be returned to the buffer specified with *buffer*. If *buflen* is less than 16 bytes, the call returns only as much data as there is room for. For example, if *buflen* = 8 bytes, the call returns only the local node name and address. Sixteen bytes are required to receive all the local node parameters.

bufoff

This integer value defines an offset, in bytes, from the address specified by the *buffer* argument. The value of *bufoff* is added to *buffer* to define the beginning address of the returned data. For example, if offset = 20, the returned data begins in byte 21 of the buffer.

byte 1 20	21 (buflen+20)
offset	returned data

whole buffer

NOTE

> The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of 10. If the actual buffer, beginning at *buffer* was only 16 bytes long, the returned data would overwrite 10 bytes beyond the end of the buffer.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example shows a NTLN call used to obtain all the local node parameters. The node name is to be stored in the string variable NAME; the address, in the integer variable ADDRSS; the node alias, in the string variable ALIAS; and the default project-programmer number, in the integer PRJPRG.

```
100 MAP (RETBUF) NODDAT$ = 20% &
    \ MAP (RETBUF) NAME$=6%,ADDRSS%,ALIAS$=6%,PRJPRG%
            .
            .
            .
200 ERRVAL% = 0% &
    \ CALL NTLN BY REF(ERRVAL%,NODDAT$,LEN(NODDAT$),0%)
```

6.5 Log User Event (NTEV) — Privleged (DECnet)

The NTEV call permits a user-written program to queue an event to the system event processor for logging. Before the event is logged, it is time-stamped by the system, in the standard Network Information and Control Exchange (NICE) protocol format. The sytem manager can use the normal network management SET and CLEAR LOGGING commands to control the filtering of these events. Optional data can accompany a user event.

NOTE

> This system call performs a highly specialized function which requires a great deal of special knowledge of DECnet and the DIGITAL Network Architecture (DNA). It is provided as a convenience for the sophisticated network user and is not intended for normal network programs.

BASIC–PLUS–2 Call Format:

CALL NTEV BY REF(*errval%,evmod%,evcls%,evtyp%,entyp%,enval%,*
buffer$,buflen%,bufoff%)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTEV call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

evmod

This integer argument specifies what the system should do if it is unable to log the user event due to lack of resources. If *evmod* is set to zero, the call terminates with a status return of NOROOM (*errval* = 4); the program can retry logging the event after a short delay. If *evmod* = 1, the system logs a "missed event" event message, and the call terminates with a successful status (*errval* = 0).

Note that in the second case, the user data is *not* logged. The system merely logs the fact that it has lost some unidentified data. This is identical to the procedure followed with DECnet internal events when the event queue is full or there are no more small buffers.

evcls

This integer argument specifies the class of the user event to be logged. Events are divided into classes, according to the DNA layer from which they originate. Classes 480 to 511 are reserved for customer-specific events. However, DECnet/E supports only class 480. Thus, *evcls* must be 480.

evtyp

This integer argument specifies the type of event within the class specified by *evcls*. The event type must be in the range of 0 to 31.

entyp

This integer argument specifies the entity type with which the event is concerned. There are four entity types supported by DECnet/E, defined as follows:

-1	No specific entity
0	Node
1	Line
3	Circuit

enval

This integer argument associates a value to the entity specified by the *entyp* argument. If the entity is a node, *enval* should be set to the node address. If the entity is a line or a circuit, *enval* should be set to the address of the Device Data Block (DDB) maintained for the device.

buffer

This argument defines the starting address of the buffer containing the optional user data to be processed with the event . If a nonzero offset is specified (see *bufoff*), the offset is added to the starting address of the buffer.

If this data is included, it must be in NICE protocol format. Basically, this means that the data should be formatted in a 3-part field: a parameter type, a data type, and a parameter value. DECnet parameter types can be used if the event concerns a node, a circuit, or a line. In this case, the event logger will output standard descriptive text for the specific parameter. If a DECnet parameter type is not used, parameter types in the range of 3900 to 4095 can be specified. (See the *DIGITAL Network Architecture, Network Management Functional Specifications*, Version 3.0.0, for further information on event definitions and parameters.)

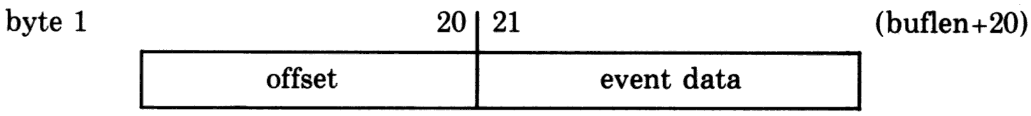
Up to 200 bytes of data can be passed to the event logger for processing.

buflen

This integer value defines the number of bytes of data in the buffer specified with *buffer*. Up to 200 bytes of data can be passed. If no data is specified, *buflen* must be 0.

bufoff

This integer value defines an offset in bytes, from the address specified in *buffer*. The value of *bufoff* is added to *buffer* to define the beginning address of the returned data. For example, if *bufoff* = 20, the data to be processed begins in byte 21 of the buffer.



whole buffer

NOTE

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes of data would be processed, starting with the byte at *buffer*-100.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
4	NO ROOM FOR USER ON DEVICE The event queue is full or there are no small buffers available to hold the event. A retry may succeed. (This is returned only if *evmod* = 0.)
5	CAN'T FIND FILE OR ACCOUNT The event logger is not running. Have the system manager start the event logger.
10	PROTECTION VIOLATION The calling program must be privileged at the time the NTEV call is issued.
18	ILLEGAL SYS() USAGE The parameters passed are either inconsistent or invalid.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example illustrates the logging of a user event, class 480. and type 3. The entity is node number 135. The optional data passed for processing consists of an *lla* parameter number (2130) and value. A "missed event" is logged if the event itself cannot be.

```
100 MAP (BUFFER) EVTDAT$ = 200% &
    \ MAP (BUFFER) PARMTY%,DATTYP$=1%,DATVAL$=197%
                 .
                 .
                 .
200 EVMOD% = 1% &
    \ EVCLS% = 480% &
    \ EVTYP% = 3% &
    \ ENTYP% = 0% &
    \ ENVAL% = 135%
                 .
                 .
                 .
300 PARMTY% = 2130% &
    \ DATTYP$ = CHR$(2%) &
    \ PVAL% = 40960% &
    \ DATVAL$ = CHR$(PVAL% AND 255%)+CHR$(SWAP%(PVAL%) AND 255%)
                 .
                 .
                 .
400 ERRVAL% = 0% &
    \ CALL NTEV BY REF(ERRVAL%,EVMOD%,EVCLS%,EVTYP%,ENTYP%, &
              ENVAL%,EVTDAT$,LEN(EVTDAT$),0%)
```

The NTEV call in the previous example generates the following event message:

```
Event type 480.3
Occurred 14-Dec-81 15:31:39.7 on node 135 (GROK)
Node address 135 (GROK)
Parameter #2130 = 40960
```

6.6 Send Calls

There are nine separate send calls — one local and eight network. Each of these calls causes NSP to format a message that is then either queued to a local program or transmitted across the network to a remote program.

6.6.1 Send Local Data Message (MSLD) (Local)

The MSLD call sends up to 532 bytes of user data to another program on the local RSTS system. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, the calling program must be privileged or the MSLD will fail with an error. The receiving program is identified in the MSLD call either by its job number or by the logical name with which it declared itself a receiver.

BASIC–PLUS–2 Call Format:

CALL MSLD BY REF(*erryal%,jobx2%,name$,buffer?,buflen%,*
bufoff%,parptr?,parlen%)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MSLD call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

jobx2

If *jobx2* is zero, the message data is sent to the local program identified by the name argument. Otherwise, the message data is sent to the program whose job number (times two) is equal to the value of *jobx2*. For Example, if *jobx2* = 8, the message is sent to job 4.

name

The logical *name* identifies the intended local receiver when the *jobx2* argument = 0. The receiver must have declared its identity with this name in its Declare Receiver call. It is a string consisting of one to six characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underline). If less than six characters are used, the name must be left-justified in the string and the remaining bytes must be padded with spaces. (A store into a 6-byte string defined in COMMON or MAP will do this automatically.) Leading or embedded spaces are invalid. This field is ignored when *jobx2* ≠ 0.

buffer

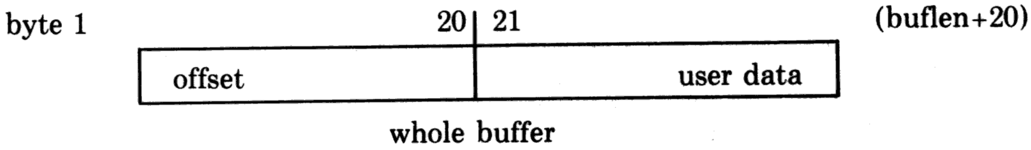
This argument defines the starting address of a buffer from which message data is to be sent. The data is assumed to be in contiguous bytes in the buffer. (See Section 6.1.1 for a discussion of different ways to set up buffers for message services in BASIC–PLUS–2.) If a nonzero offset is given (see *bufoff*) the offset is added to the starting address specified by *buffer*.

buflen

This integer argument defines the number of bytes to be sent from the buffer. Possible values for *buflen* range from 0 through 512.

bufoff

This integer value defines an offset, in bytes, from the address specified by the buffer argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address of the message data to be sent. For example, if offset = 20, the user data begins in byte 21 of the buffer.



NOTE

> The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512 and a *bufoff* of 1000. The MSLD call would send 512 bytes beginning at *buffer*+1000.

parptr

This argument defines an area containing up to 20 bytes of parameter message data to be sent to the local program. This data will be delivered separate from the message data in the buffer area. For example, a RSTS BASIC-PLUS-2 program issuing a Receive call that returns a Local Data Message will get the parameter data and the message data in two separate buffers defined in the Receive call.

parlen

This integer value defines the number of bytes to be sent from the parameter area defined by *parptr*. The *parlen* value can range from 0 through 20.

Possible Errors:

errval	**Error Text and Meaning**
4	NO ROOM FOR USER ON DEVICE The number of pending messages for the intended local receiver is at its declared maximum. The program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	CAN'T FIND FILE OR ACCOUNT The intended local receiver could not be located in the system's list of declared receivers. The receiver must be declared (with a Declare Receiver call) before any data can be transmitted to it.
10	PROTECTION VIOLATION Some access violation has occurred. Either the receiver does not allow any local senders or the sender is nonprivileged and the receiver allows only privileged senders.

(continued on next page)

errval	**Error Text and Meaning**
18	ILLEGAL SYS() USAGE The value of *jobx2* is odd. It must be 0 or the receiver's job number times two.
31	ILLEGAL BYTE COUNT FOR I/O Either the *buflen* value or the *parlen* value is invalid. The *buflen* argument can range from 0 through 512. The *parlen* argument can range from 0 through 20.
32	NO BUFFER SPACE AVAILABLE System buffers are currently not available to store this message for the intended local receiver. A later retry may proceed without error.

Example:

The following example shows a local send to a program that has issued a Declare Receiver call with the logical name LOCAL. A buffer length (*buflen* argument) of 8 and offset of 18 indicate that the characters 'OFFSET' are sent. No parameter message data is sent.

```
1000  COM A$ = 512% &
      \COM LNAME$ =6% &
      \ERRVAL% = 0% &
      \A$ = "TO ILLUSTRATE THE "OFFSET" &
      \LNAME$ = "LOCAL" &
      \BUFLEN% = 8% &
      \BUFOFF% = 18% &
      \CALL MSLD BY REF(ERRVAL%,0%,LNAME$,A$,BUFLEN%, &
       BUFOFF%,0%,0%)
```

6.6.2 Send Connect Initiate Message (NTCI) (DECnet)

The NTCI call is used to request a logical link to another program. The other program is identified with a Connect Data Block and the user link address by which the calling program will refer to this link is established. Other arguments specify the flow control option and maximum amount of user data for received Network Data Messages. Up to 16 bytes of user data can be sent with the NTCI call.

BASIC-PLUS-2 Call Format:

CALL NTCI BY REF(*erral%,ula%,llmod%,rmax%,buffer?,*
buflen%,bufoff%)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTCI call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The user link address (*ula*) is an integer value within the range 1 to 255. Later calls to send and receive messages use this value to refer to the logical link (assuming that the link being requested by the NTCI is accepted by the remote NSP and the remote program).

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

llmod

The integer value *llmod* indicates the type of flow control requested by the calling program to control incoming data from the remote program. Acceptable values and meanings are:

0 No flow control
1 Segment flow control
2 Message flow control

Flow control is described in detail in Chapter 4. Remember that each program in a logical link selects its own flow control. The *llmod* value is the selection made by the local program. If *llmod* is nonzero, the program must issue Link Service (NTLS) calls to request Network Data Messages from the other program.

rmax

A program that has limited buffer space and is not designed to process large messages in small pieces, can impose a limit on the amount of user message data it is willing to receive on a logical link. The integer value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by NSP will itself limit the amount of data that can be passed in a single Network Data Message. If the maximum size set for NSP by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local NSP will alter the maximum to the smaller limit before forwarding the Connect Initiate Message to the remote node. If *rmax* = 0, NSP will supply the size of its receive buffers as a default value. (The calling program will be informed of any such change by a field in the Connect Confirm Message from the target program when the link is accepted.)

The remote NSP uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument with the Receive call (MRCV) in Section 6.7 for a description of how to process large messages in small pieces.

buffer

This argument defines the starting address of a buffer for the Connect Data Block and optional user message data. The 120-byte Connect Data Block and up to 16 bytes of message data are assumed to be in the buffer in contiguous bytes. (See Section 6.1.1 for information on how to set up a buffer in BASIC-PLUS-2.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

The Connect Data Block identifies the target program for the requested logical link. Figure 6–1 shows the format of the Connect Data Block. The fields within the block are described in Table 6–3.

Up to 16 bytes of user data can follow the Connect Data Block.

buflen

This integer value defines the number of bytes to be sent. Since the Connect Data Block is 120 bytes long and the optional user data can be from 0 to 16 bytes long, acceptable values for *buflen* range from 120 through 136.

bufoff

This integer value defines an offset, in bytes, from the start of the buffer from which the data is to be sent. The offset value is added to the starting address of the buffer to define where the Connect Data Block begins. For example, if *bufoff* = 20, the Connect Data Block begins in byte 21 of the buffer.

byte 1 … 20	21 … 140	141
offset	Connect Data Block	user data

whole buffer

NOTE

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. If a program's buffer is 100 bytes long, for example, with a specified offset of 20 and a length of 136, 136 bytes will be sent: the last 80 bytes of the buffer and 56 bytes of whatever is beyond the buffer in memory.

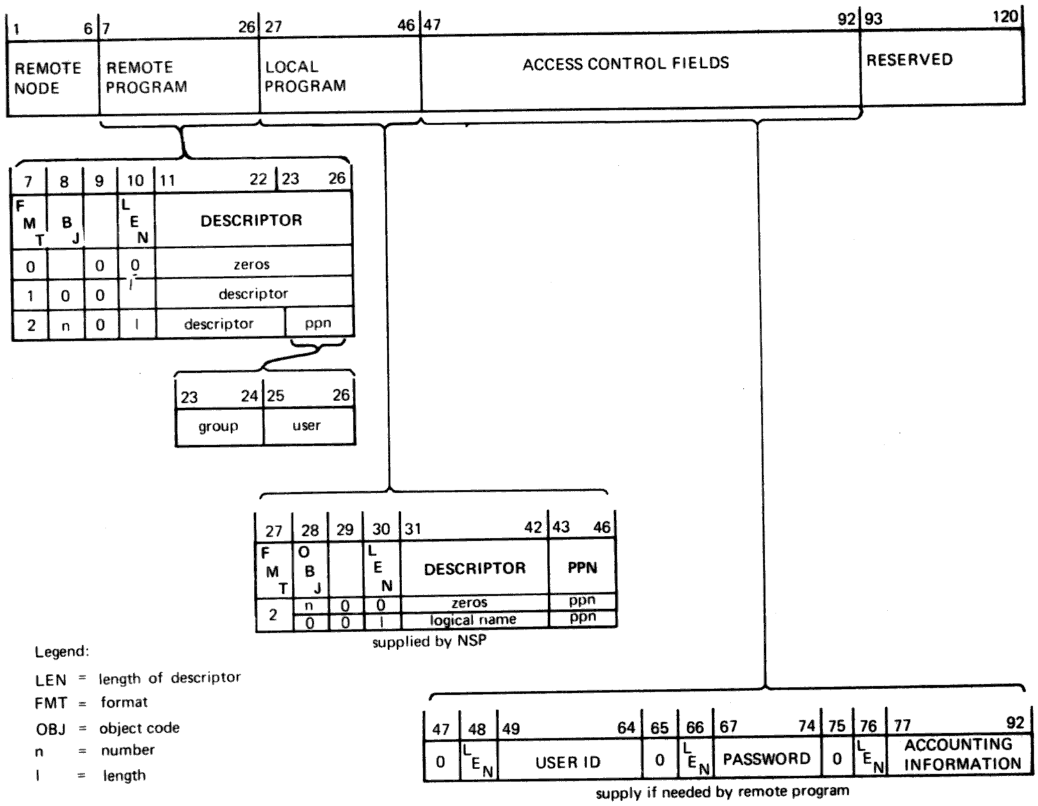


Figure 6–1: Format of Connect Data Block on Send Connect Initiate Message

Table 6–3: Format of Connect Data Block on Send Connect Initiate Message

Bytes	Content
1–6	**Remote Node**. 1 to 6 uppercase, alphanumeric ASCII characters naming the remote node to which the Connect Initiate Message is to be sent. The name must contain at least one alphabetic character, and names shorter than six characters must be left-justified and padded to six characters with spaces. If no node name is to be specified (that is, the request is directed to a program at the local node), this field should contain 6 spaces.
7–26	**Remote Program**. Identifies the remote program to which the Connect Initiate Message is directed as either an object type (general function) or a specific entity (program, task, or process) This identification is given in one of three formats:

Format 0

The remote program is identified by object type code alone. The fields are:

Bytes	Content
7	Zero, to indicate a format 0 name.
8	Object type code of the remote program.
9–26	Zero (Not used).

(continued on next page)

Table 6–3 (Cont.): Format of Connect Data Block on Send Connect Initiate Message

Format 1

The remote program is identified by a descriptor. The fields are:

Bytes	Content
7	One, to indicate a format 1 name.
8	Normally zero. This byte should be zero unless the target DECnet system allows identification by both name and object type code on incoming connect requests (DECnet does not). If nonzero, this byte is interpreted as the object type code of the remote program for which the Connect Initiate Message is intended.
9	Reserved - must be zero.
10	Length of descriptor, 0 to 16 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field.
11–26	Remote program descriptor (left-justified). For a DECnet program, this is the logical name that the remote program defined in its Declare Receiver call. For other systems, it is the descriptor used to register the program for network operations.

Format 2

This format allows identification of the remote program by name or object type code and by the group and user codes under which the remote program is accessible on the remote system. (The group and user codes refer to what is called the project-programmer number on some DIGITAL systems, including RSTS. Other systems refer to these codes as the user identification code, or UIC.) DECnet systems use only the object type code or descriptor to find or start the target program for which a Connect Initiate Message is intended. The group and user codes are simply passed on to the target program for its own checking. Other DECnet systems can handle an incoming connect request in format 2 differently.

The subfields are defined as follows:

Bytes	Content
7	Two, to indicate a format 2 name.
8	Object type code of the remote program, if applicable. If zero, the descriptor must be used.
9	Reserved - must be zero.
10	Length of descriptor, 0 to 12 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11–22).
11–22	Remote program descriptor, if this is used instead of object type code. The descriptor should be left-justified in the field.
23–24	Remote group code. Binary value giving the group number under which the remote program is running or is to be started. Corresponds to the project number in a RSTS project-programmer number. This value is not used by DECnet systems but is simply passed on to the receiving program.
25–26	Remote user code. Binary value giving the user number under which the remote program is running or is to be started. Corresponds to the programmer number in a RSTS project-programmer number. This value is not used by DECnet systems but is simply passed on to the receiving program.

27–46 **Local Program**. This field is filled in by the local NSP using information that the local program specified in its Declare Receiver call. Any values passed by the user program in this area are ignored.

(continued on next page)

Table 6-3 (Cont.): Format of Connect Data Block on Send Connect Initiate Message

	The information is passed to the remote system in format 2, as described previously. Byte 27 is filled in with a value of 2. Byte 29 is always zero. If the program declared its identity with object type alone, the object type code is passed in byte 28 and bytes 30–42 are zero. If the program declared its identity with a logical name, or with a logical name and object type code, the local name is supplied in bytes 31–42 and byte 28 is zero. The length of the logical name is given in byte 30. The project-programmer number under which the program is running is passed in bytes 43–46 as the group and user codes. The project number is in byte 43. The programmer number is in byte 45. Both are in binary format.
47–92	**Access Control Fields**. These bytes can be used to define the calling program's access rights to the remote system or to the remote system's services. The idea is analogous to logging in on the remote system. The most rigorous checking done in a normal log-in on DIGITAL systems involve user identification, password, and account number, so three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all nor does it require that the fields be used as described. A DECnet system simply passes this information on to the receiving program. To determine what, if anything, a non-RSTS DECnet system requires in these fields, see the DECnet manual for that system.

NOTE

The DECnet utilities NFT and NETCPY, that use logical links to access remote files and devices, use these fields to establish the local terminal user's right to access remote files and devices. Both utilities prompt the local user for log-in information to be used at the remote system and pass that information along in the Connect Data Block of the Connect Initiate Message. The remote FAL or receiving NETCPY checks this information before allowing the operation requested.

Bytes	**Content**
47	Reserved - must be zero.
48	Length of user identification information specified in the next field, 0 to 16 bytes.
49–64	User Identification. This field identifies the person or program requesting the logical link. In RSTS terms, the user ID is analogous to the project-programmer number used at log-in. This information need only be specified here if it is required by the remote system or program. If it is not required, the length (byte 48) should be set to zero.
65	Reserved - must be zero.
66	Length of password information specified in the next field, 0 to 8 bytes.
67–74	Password. A password is often used to verify access to a system under a particular user identification. In RSTS, this password is analogous to the password used in conjunction with the project-programmer number at log-in. A password supplied here must be acceptable to the remote system or program, if one is required. If not required, the length (byte 66) should be set to zero.
75	Reserved - must be zero.
76	Length of the accounting information specified in the following field, 0 to 16 bytes.
77–92	Accounting information. Many systems require a billing account number in addition to user identification and password. The accounting field is provided for this purpose. Once again, such information need only be specified here if it is required by the remote system or program. If it is not required, the length (byte 76) should be set to zero.

(continued on next page)

Table 6–3 (Cont.): Format of Connect Data Block on Send Connect Initiate Message

93–120 **Reserved for future use.** Currently ignored by DECnet but should be passed as zeros.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
3	ACCOUNT OR DEVICE IN USE The calling program already has a logical link with the same user link address (*ula*) as specified in this Connect Initiate Message. A different *ula* must be used or the existing link must be disconnected.
6	NOT A VALID DEVICE The node named in the Connect Data Block is not known to the system.
14	DEVICE HUNG OR WRITE LOCKED The node named in the Connect Data Block is known to the system but is currently inactive. There is no physical communication path to the node.
17	TOO MANY OPEN FILES ON UNIT The total number of logical links allowed for the local node has been reached. A later try may succeed. (The system manager can set a limit on the total number of links. If this limit is inadequate, see the system manager.)
18	ILLEGAL SYS() USAGE One of several possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The Declare Receiver call for the calling program had a null logical name and a zero object type code. There is no way to identify the program to the remote system. Issue a Remove Receiver call and reissue the Declare Receiver call. 3. An invalid value was detected in one of the following fields. • Remote program field in the Connect Data Block. • One of the three access control fields in the Connect Data Block. • The user link address (*ula*) given was zero. Valid values range from 1 to 255. • The local link modifier (*llmod*) value must be 0, 1, or 2. • The remote descriptor length (byte 10 in the Connect Data Block) was nonzero but the descriptor field was null (all zeros).

(continued on next page)

errval	**Error Text and Meaning**
22	DISK PACK IS LOCKED OUT NSP cannot create a new logical link because the physical line to the remote node or the local node itself has been scheduled for shutdown by the system manager.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 120 bytes and 136 bytes for a Connect Initiate Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example shows a Connect Initiate Message directed to a program named NETRCV at node MIAMI. The Connect Data Block uses a format 1 name to address the remote program. (Note the use of the SWAP% function to place a value of 6 in byte 10 of the Connect Data Block.) NSP supplies the local program identification and no access control fields are used. Thus, the rest of the Connect Data Block is filled with zeros. Message data is sent following the Connect Data Block. Note that 16 bytes of user message data are actually sent. Storing into a string field defined by MAP causes space fill in the field. Thus, the LEN function will return a value of 136.

```
1000   MAP (SNDCI) CONDAT$ = 136% &
       \MAP (SNDCI) NODE$ = 6%,FMT%,LGT%,NAME$=16%, &
             FILL$(94%)=CHR$(0%),MSG$=16% &
                 .
                 .
                 .
       \ULA% = 27% &
       \LLMOD% = 2% &
       \RMAX% = 512% &
       \NODE$ = "MIAMI" &
       \FMT% = 1% &
       \LGT% = SWAP%(6%) &
       \NAME$ = "NETRCV" &
       \MSG$ = "OPTION2" &
                 .
                 .
                 .
       \CALL NTCI BY REF(ERRVAL%,ULA%,LLMOD%,RMAX%, &
            CONDAT$,LEN(CONDAT$),0%)
```

6.6.3 Send Connect Confirm Message (NTCC) (DECnet)

The NTCC call is used to accept a remote program's request for a logical link. The call establishes the user link address that the calling program will use to refer to the link. The local link address assigned by the local NSP identifies the particular link being accepted. The flow control option and the receive maximum for the link are specified and up to 16 bytes of message data can be passed to the remote program.

BASIC-PLUS-2 Call Format:

CALL NTCC BY REF(*errval%,ula%,lla%,llmod%,rmax%,*
buffer?,buflen%,bufoff%)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTCC call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The user link address (*ula*) is an integer value within the range 1 to 255. Later calls to send and receive messages use this value to refer to the logical link.

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

lla

The local link address (*lla*) is an integer value that identifies the link being accepted. It is a number assigned by the local NSP when it received the Connect Initiate Message requesting the link. The local link address is passed to the local program in bytes 5–6 of the received Connect Initiate Message. (Section 6.7.2 describes the information returned for a received Connect Initiate Message.)

llmod

The integer value *llmod* indicates the type of flow control requested for this end of the logical link. Acceptable values and meanings are:

0 No flow control
1 Segment flow control
2 Message flow control

Flow control is described in detail in Chapter 4. Remember that both programs in a logical link select their own flow control, independent of each other. The *llmod* value is the selection made by the local program. If *llmod* is nonzero, the program must issue Link Service (NTLS) calls to request Network Data Messages from the other program.

rmax

A program could have limited buffer space and not be coded to process large messages in small pieces. Such a program can impose a limit on the amount of user message data it is willing to receive on a logical link. The integer value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by NSP will itself limit the amount of data that can be passed in a single Network Data Message. If the maximum size set for NSP by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local NSP will alter the maximum to the smaller limit before forwarding the Connect Confirm Message to the remote node. If *rmax* = 0, the local NSP will supply the size of its receive buffers as a default value.

The remote NSP uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument for the Receive call (MRCV) in Section 6.7 for a description of how to process large messages in small pieces.

buffer

This argument defines the starting address of a buffer for the optional user message data. The data is assumed to be in the buffer in contiguous bytes. (See Section 6.1.1 for a discussion of how to set up buffers for message operations in BASIC-PLUS-2.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer value defines the number of bytes of user data to be sent: 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the address specified by the buffer argument. The offset value is added to the starting address of the buffer to define where the user data begins. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.

byte 1 ... 20	21 ... (buflen+20)
offset	user data

whole buffer

NOTE

> The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
3	ACCOUNT OR DEVICE IN USE A logical link is currently active for the calling program with the same user link address (*ula*) as specified in this Connect Confirm Message. A different *ula* must be used or the existing link must be disconnected.
5	CAN'T FIND FILE OR ACCOUNT The local link address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the latter case, a Link Abort Message has already been queued for this link. The *lla* field in the received Link Abort Message identifies the link, since no *ula* was ever established.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Connect Confirm Message for a link that is not waiting for confirmation will cause this error. For example, a second Connect Confirm Message for the same logical link will cause this error because the link is already established.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. An invalid value was detected in one of the following fields. • The user link address (*ula*) given was zero. Valid values range from 1 to 255. • The local link modifier (*llmod*) value must be 0, 1, or 2.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. It must be between 0 and 16 (bytes) for a Connect Confirm Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

In this example, an NTCC call responds to a received Connect Initiate Message that was stored in the receiving string REC$. The local link address is stored in bytes 5 and 6 of such a string and the second MAP statement has been used so that LLA% obtains this integer value. The call also establishes a user link address of 15, a receive maximum of 256 bytes, and indicates that the program requests message flow control. No user message data is sent, so a null dynamic string and zero values are used for the buffer arguments.

```
1000 MAP (RETBUF) REC$ = 40% &
     \MAP (RETBUF) FILL$ = 4%,LLA%,FILL%,REMAINDER%, &
          FILL%,LGT%,FILL$=8%,RLMOD$=1%,FILL$=1%, &
          RMAX%,TMAX% &
              .
              .
              .
     \ERRVAL% = 0% &
     \ULA% = 15% &
     \LLMOD% = 2% &
     \RMAX% = 256% &
     \CALL NTCC BY REF(ERRVAL%,ULA%,LLA%,LLMOD%, &
          RMAX%,NOBUF$,0%,0%)
```

6.6.4 Send Connect Reject Message (NTCR) (DECnet)

The NTCR call is used to reject a remote program's request for a logical link. The link is identified by the local link address established by the local NSP. Up to 16 bytes of user message data can be sent to the remote program as part of the Connect Reject Message.

BASIC–PLUS–2 Call Format:

CALL NTCR BY REF(*erval%,lla%,reason%,buffer?,buflen%,bufoff%*)

Argument Descriptions:

errval

The *erval* argument must be an integer variable name. If the NTCR call succeeds, this variable is set to zero. If an error occurs, *erval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

lla

The local link address (*lla*) is an integer value that identifies the link being rejected. It is a number assigned by the local NSP when it received the Connect Initiate Message requesting the link. The local link address is passed to the local program in bytes 5-6 of a received Connect Initiate Message. (Section 6.7.2 describes the information returned for a received Connect Initiate Message.)

reason

DECnet/E NSP allows one of the following three reason code values for this word: 0, 34, or 36.

Most user programs will simply use 0. If any reason is supplied for rejecting the connection, it is passed in the 16-byte user data area.

The values 34 and 36 are provided so that DECnet/E support programs (such as FAL and NFT) can reject unauthorized or improper requests for access to local RSTS/E files in a manner agreed upon as part of the overall DECnet design. There is no reason for a user program to use these codes unless it talks to programs that interpret them. All DECnet programs that are concerned with protection against unauthorized access are designed to recognize these codes. These programs take appropriate action, such as displaying an error message to a user, when they receive such a connect rejection.

The value 34 indicates that the user ID and password subfields in the Connect Data Block of the remote program's Connect Initiate Message do not correspond to any valid user known to the local system.

The value 36 indicates that, while the user ID and password subfields are acceptable, the account subfield is not. For example, the specified user may not be authorized to use that billing account or the account may be exhausted.

buffer

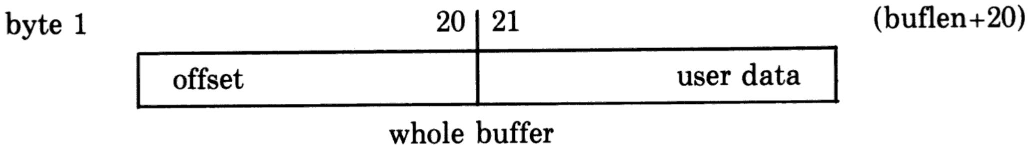
This argument defines the starting address of the buffer containing the optional user message data. The data is assumed to be in the buffer in contiguous bytes. (See Section 6.1.1 for a discussion of how to set up buffers for message operations.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer value defines the number of bytes of user data to be sent: 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the *buffer* argument. The value of *bufoff* is added to the buffer address to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



NOTE

> The BASIC–PLUS–2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
5	CAN'T FIND FILE OR ACCOUNT The local link address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the latter case, a Link Abort Message has already been queued for this link. The *lla* field in the received Link Abort Message identifies the link, since no *ula* was ever established.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Connect Reject Message for a link that is not waiting for confirmation will cause this error. For example, sending a Connect Reject Message for a link that is already established will return this error.

(continued on next page)

errval	**Error Text and Meaning**
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The reason code supplied in the Connect Reject Message call was not 0, 34, or 36.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 bytes for a Connect Reject Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

In this example, a NTCR call responds to a received Connect Initiate Message stored in the receiving buffer R$. A reason code of 0 is supplied. The local program notifies the remote program of the reason for rejection with user data.

```
1000 MAP (RETBUF) R$ = 40% &
     \MAP (RETBUF) FILL$ = 4%,LLA%,FILL%,REMAINDER%,FILL%, &
          LGT%,FILL$=8%,RLMOD$=1%,FILL$=1%,RMAX%,TMAX% &
     \ERRVAL% = 0% &
     \BUFFER$ = "TOO LATE" &
     \BUFLEN% = 8% &
     \BUFOFF% = 0% &
     \REASON% = 0% &
             .
             .
             .
     \CALL NTCR BY REF(ERRVAL%,LLA%,REASON%,BUFFER$, &
          BUFLEN%,BUFOFF%)
```

6.6.5 Send Network Data Message (NTDM) (DECnet)

The NTDM call transmits user data — a *segment*, as defined in Chapter 4 — over an established logical link. A flag byte indicates whether or not the segment is the end of a logical message. If the remote program requested segment or message flow control when the link was established, requests for data must be outstanding or this call will result in an error.

BASIC–PLUS–2 Call Format:

CALL NTDM BY REF(*errval%,retbuf?,ula%,dmflgs%,buffer?,*
buflen%,bufoff%)

Argument Descriptions:

errval

The *erroal* argument must be an integer variable name. This variable is set to zero if the NTDM call succeeds. If an error occurs, *erroal* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the buffer to receive link status information returned by the call. This argument must be specified regardless of whether or not link status information is requested and should refer to a preallocated buffer at least 40 bytes long.

ula

The *ula* is an integer value giving the user link address (1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

dmflgs

The *dmflgs* argument can be coded as the sum of three bit values:

dmflgs% = *rls%* + *eom%* + *bom%*

The low-order two bits of the *dmflgs* argument (*eom%* + *bom%*) indicate whether the segment being sent in this Network Data Message is the first, last, middle, or sole segment of a logical message. If message flow control was requested by the remote program when the link was established, NSP uses these two bits to regulate the transmission of logical messages as the remote program requests them. (See Chapter 4 for a full discussion of flow control.)

eom	*bom*	
0	0	Middle segment of a logical message
0	1	First segment of a logical message
2	0	Last segment of a logical message
2	1	Sole segment of a logical message

DECnet/E NSP uses the *eom* bit to count logical messages transmitted to the remote program. The *bom* bit is not used in maintaining the logical message request count but is simply passed to the remote system and program.

NSP also uses the *eom* bit to determine whether it will permit a disconnect of the logical link. If a disconnect is requested but the last Network Data Message sent did not have the *eom* bit set, NSP assumes that the last message has not been completely sent and will reject a Disconnect Message. This is always done, irrespective of the flow control option selected by the remote program.

Some DECnet implementations require that the *eom* flag be set to terminate an asynchronous I/O operation. Thus, when the remote program requests segment flow control or no flow control, the *eom* flag should always be set. Setting the bit in this case will not affect a remote DECnet/E node but is essential if the remote node is a DECnet node waiting to terminate asynchronous I/O.

The *rls* bit is used to request that link status information be returned to the *retbuf* argument when this call is successfully completed.

rls = 128	Link status information is returned.
rls = 0	No link status information is returned.

If link status information is requested, it is returned in the same format as a received Link Service Message (Section 6.7.7). The MAP statement is particularly useful to allocate the buffer, as it both allocates the space for the information and allows the definition of named fields with different data types within the buffer (see Section 6.1.1).

If no link status information is requested, random data is returned.

buffer

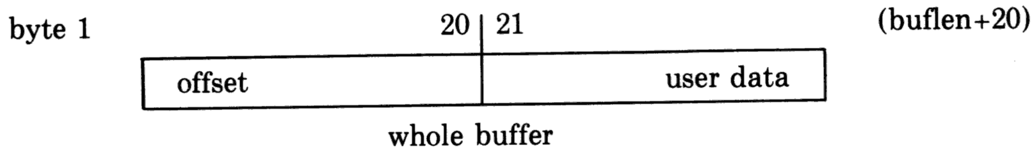
This argument defines the starting address of a buffer containing the user message data. The data is assumed to be in contiguous bytes. (See Section 6.1.1 for a discussion of how to set up buffers in BASIC-PLUS-2.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer value defines the number of bytes to be sent. The maximum amount of data that can be transferred over the logical link is established by the remote program or remote NSP. It is passed on to the local program as the transmit maximum — the *tmax* value in a received Connect Initiate or Connect Confirm Message (Sections 6.7.2 and 6.7.3). Thus, the value of *buflen* can range from 1 through *tmax*.

bufoff

This integer value defines an offset, in bytes, from the starting address of the buffer defined by the buffer argument. The value of *bufoff* is added to the buffer address to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



NOTE

> The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. One hundred bytes beginning at *buffer*+20 would be sent.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
4	NO ROOM FOR USER ON DEVICE The Data Message transmit queue for this logical link is full. NSP is waiting for acknowledgment from the remote system for previously sent Network Data Messages. No more can be sent until at least or of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should simply retry the send after a short (1 to 5 second) delay. (See Chapter 4 for a full discussion of flow control.)
5	CAN'T FIND FILE OR ACCOUNT The user link address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Network Data Message over a logical link that has not yet been confirmed will cause this error.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The *ula* specified is zero. It must be within the range 1 to 255.

(continued on next page)

errval	**Error Text and Meaning**
19	DISK BLOCK IS INTERLOCKED Either the remote system has inhibited transmission on this link off because of a backpressure condition or there are no outstanding requests for segments or logical messages from the remote program (assuming it requested segment or message flow control for the link). The program will be notified with a Link Service Message when the condition clears, as described in Chapter 4.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 1 and the transmit maximum (*tmax*) established by the remote program or system in its Connect Initiate or Connect Confirm Message for this logical link.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example shows a variable-length send from a buffer area defined by a dynamic string. The LEN function returns the correct number of bytes for the length argument. The user link address is 20. The value for DMFLGS% identifies the message data as the sole segment of a logical message and requests the return of link status information. Note the use of MAP statements to allocate space for the buffer for the returned data. First, the buffer for returned data is set up as 40 bytes long. Next, a MAP is made with a useful format for returned link status information. (Other maps of the same area would presumably be made for other message types. They are not shown here since we are concerned only with returned link status.) After the call, the program checks for an error. If none occurred, the program checks to see if there are any pending messages. If so, the program branches to receive and process messages. If not, the program simply continues. Note the use of the ASCII function to get integer values for the returned data.

```
1000 MAP (RETBUF) A$ = 40% &
     \MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=16%, &
          LSTAT$=1%,RSTAT$=1%,LDR$=1%,&
          LIR$=1%,RDR$=1%,RIR$=1%,DTM$=1%,DTC$=1%,LTM$=1%, &
          LTC$=1%,MMAX$=1%,MCNT$=1%,MULA$=1% &
     \EOM% = 2% &
     \BOM% = 1% &
     \RLS% = 128% &
                .
                .
                .
     \B$ = "DYNAMIC STRING SEND - LEN WILL RETURN LENGTH" &
     \ERRVAL% = 0% &
     \ULA% = 20% &
     \DMFLGS% = EOM%+BOM%+RLS% &
     \CALL NTDM BY REF(ERRVAL%,A$,ULA%,DMFLGS%,B$, &
          LEN(B$),0%) &
     \IF ERRVAL% <> 0% THEN GOTO 1500 &
     \IF ASCII(MCNT$) > 0 THEN GOTO 4000 &
                .
                .
                .
```

6.6.6 Send Interrupt Message (NTIN) (DECnet)

The NTIN call transmits an Interrupt Message to a remote program over an established logical link. All DECnet systems are designed to deliver Interrupt Messages ahead of other messages. If the remote system is a DECnet/E system, the Interrupt Message will be placed at the head of the pending message queue, behind the first message (since it can already be partly received) and behind any other pending Interrupt Messages queued for the program. Interrupt Messages are subject to flow control, as described in Chapter 4.

BASIC–PLUS–2 Call Format:

CALL NTIN BY REF(*erval%,retbuf?,ula%,inflgs%,buffer?,*
buflen%,bufoff%)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTIN call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the buffer to receive link status information returned by the call. This argument must be specified regardless of whether or not link status information is requested and should refer to a preallocated buffer at least 40 bytes long.

ula

The *ula* is an integer value giving the user link address (1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

inflgs

The *inflgs* argument indicates whether link status information is to be returned to *retbuf* when the call has completed successfully.

inflgs% = *rls%*

where

rls = 128 Link status information is returned.
rls = 0 No link status information is returned.

If link status information is requested, it is returned in the same format as a received Link Service Message (Section 6.7.7). The MAP statement is particularly useful to allocate the buffer, as it both allocates the space for the information and allows the definition of named fields with different data types within the buffer (see Section 6.1.1).

If no link status information is requested, random data is returned.

buffer

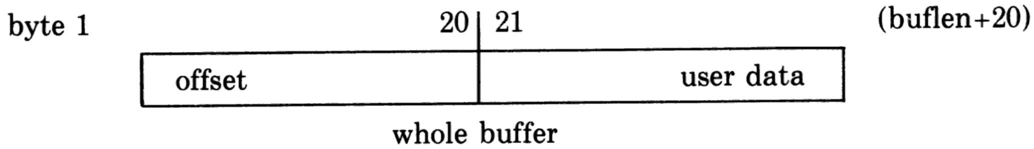
This argument defines the starting address of a buffer containing optional user message data. (See Section 6.1.1 for information on how to set up a buffer in BASIC–PLUS–2.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer to determine where the data begins.

buflen

This integer value defines the number of bytes of user data to be sent: 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the *buffer* argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



NOTE

> The BASIC–PLUS–2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. NSP is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should simply retry the send after a short (1 to 5 second) delay. (See Chapter 4 for a full discussion of flow control.)
5	CAN'T FIND FILE OR ACCOUNT The local link address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.

(continued on next page)

errval	**Error Text and Meaning**
10	PROTECTION VIOLATION Some procedural error has occurred. Sending an Interrupt Message over a logical link that has not yet been confirmed will cause this error.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The user link address (*ula*) given was zero. Valid values range from 1 to 255.
19	DISK BLOCK IS INTERLOCKED The remote NSP has prohibited transmission of Interrupt Messages over this logical link according to its rules for Interrupt Message flow control. A link Service Message will be delivered to the program when the condition clears. (See Chapter 4 for a full discussion of flow control.)
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 bytes for an Interrupt.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example shows a NTIN call for logical link 15. The call sends the message data "TIMEOUT" and requests no link status information to be returned. A buffer must be defined and referenced, however, so a MAP is set up with the string name DUMMY$ for reference in this case. (The same buffer area can be used with different mappings for other calls. The name DUMMY$ documents that the data returned is random and not used by the program.)

```
1000 MAP (RETBUF) DUMMY$ = 40% &
                .
                .
                .
     \ERRVAL% = 0% &
     \A$ = "TIMEOUT" &
     \BUFLEN% = LEN(A$) &
     \OFFSET% = 0% &
     \INFLGS = 0% &
     \ULA% = 15% &
     \CALL NTIN BY REF(ERRVAL%,DUMMY$,ULA%,INFLGS%, &
       A$,BUFLEN%,BUFOFF%)
```

6.6.7 Send Link Service Message (NTLS) (DECnet)

The NTLS call is used to: (1) request data from a remote program over a flow-controlled logical link (Chapter 4); (2) reenable incoming interrupts over a logical link (Chapter 4); or (3) obtain status information for the link. No user data is allowed with a Link Service Message.

BASIC–PLUS–2 Call Format:

CALL NTLS BY REF(*erryal%,retbuf?,ula%,lsflgs%,drcnt%,ircnt%*)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTLS call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the buffer to receive link status information returned by the call. The argument must be specified regardless of whether or not link status information is requested and should refer to a preallocated buffer at least 40 bytes long.

ula

The *ula* is an integer value giving the user link address (1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

lsflgs

The link service flags argument is coded as the sum of two values: *lsflgs%* = *rls%* + *opr%*. The low-order two bits (*opr*) indicate the purpose of the call.

opr = 0 Indicates the call is requesting segments or logical messages from the remote program. In this case, the *drcnt* argument is interpreted as the number of segments or logical messages being requested. This form of the call can be used only when the calling program requested message or segment flow control in its Connect Initiate or Connect Confirm Message for the logical link. If no flow control was requested, an NTLS call with *opr* = 0 will fail, returning *errval* = 18.

= 1 Indicates the call is to reenable incoming Interrupt Messages for this link. In this case, the *ircnt* argument is interpreted as the number of Interrupt Messages requested (always 1).

= 2 Indicates the call is to obtain status information only. In this case, the *drcnt* and *ircnt* arguments are ignored.

The *rls* bit can be used when *opr* = 0 or 1 to request that link status information be returned to the return buffer (defined by *retbuf*) when the call completes successfully.

rls = 128 Link status information is returned.
rls = 0 No link status information is returned.

Link status information is always returned when *opr* = 2. If requested, the information returned is in the same format as a received Link Service Message (Section 6.7.7) The MAP statement is particularly useful to define the buffer, as it both allocates the space for the information and allows the definition of named fields with different data types within the buffer (see Section 6.1.1).

If no link status information is requested, random data is returned.

drcnt

The data request count (*drcnt*) is meaningful only when *opr* = 0 and indicates the maximum number of segments or logical messages being requested from the remote program. Depending on the flow control option in effect for the logical link, the value is interpretted as either a segment counter (segment flow control) or a logical message counter (message flow control). In either case, it is an incremental count, added to the counter of outstanding requests kept by the remote and local NSP (see Chapter 4). An error is returned if the *drcnt* given increases the counter to more than 127.

ircnt

The interrupt request count (*ircnt*) is meaningful only when *opr* = 1 and indicates the number of Interrupt Messages being requested. DECnet/E allows only one interrupt request outstanding for a logical link at any given time. Hence, *ircnt* must always be 1 when *opr* = 1 and has the effect of reenabling interrupts on the logical link. Remember that interrupts can be reenabled only when the previous Interrupt Message for this logical link, if any, has been received from the pending message queue. If it has not, the call will fail with an error (*errval* = 19).

Possible Errors:

errval	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. NSP is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent

(continued on next page)

errval **Error Text and Meaning**

until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and will not cause a Link Service Message to be queued when the condition clears. The program should simply retry the send after a short delay of 1 to 5 seconds. (See Chapter 4 for a full discussion of flow control.)

5 CAN'T FIND FILE OR ACCOUNT

The user link address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.

10 PROTECTION VIOLATION

Some procedural error has occurred. Sending a Link Service Message over a logical link that has not yet been confirmed will cause this error.

18 ILLEGAL SYS() USAGE

One of several possibilities:

1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given.
2. The *opr* value is not 0, 1, or 2.
3. The *opr* value is 0, but the calling program did not request flow control for the logical link in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message.
4. The data request count (*drcnt*) increased the count of outstanding requests to more than 127.
5. The *opr* value is 1, indicating a request for an Interrupt Message, but the *ircnt* argument does not equal 1.
6. The *opr* value is 1, indicating a request for an Interrupt Message, but the outstanding requests counter for interrupts already equals 1.
7. The *ula* specified is zero. It must be within the range 1 to 255.

19 DISK BLOCK IS INTERLOCKED

The *opr* value is 1, indicating a request for an Interrupt Message, but there is already one queued for this logical link. See Chapter 4 for details on flow control.

32 NO BUFFER SPACE AVAILABLE

System buffers are not currently available to store this message. A later try may succeed.

62 NO RUN-TIME SYSTEM

NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.

66 MISSING SPECIAL FEATURE

DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

This example of the NTLS call requests five segments from the remote program over logical link 23. (Assume that the local program requested segment flow control in its NTCI or NTCC call for this link).

```
1000 MAP (RETBUF) A$ = 40% &
     \OPR% = 0% &
     \RLS% = 128% &
            .
            .
            .
     \ERRVAL% = 0% &
     \ULA% = 23% &
     \LSFLGS% = RLS%+OPR% &
     \DRCNT% = 5% &
     \CALL NTLS BY REF(ERRVAL%,A$,ULA%,LSFLGS%,DRCNT%,0%) &
```

6.6.8 Send Disconnect Message (NTDI) (DECnet)

The NTDI call is used to terminate an established logical link. Messages in the pending message queue will remain but no more messages can be sent over the link. The user link address is freed and can be used for another logical link. If the last Network Data Message sent did not have the end-of-message flag set, or if any Network Data, Interrupt, or Link Service Messages are still waiting for acknowledgment in the transmit queues for this logical link, this call will fail with an error.

Successful completion of an NTDI assures the sender that the remote system has received and acknowledged all Network Data, Interrupt, and Link Service Messages previously sent over the link. It does not guarantee that the receiving program has processed the messages, however. The Disconnect Message is useful in "master-slave" communication where the master program only transmits data and the slave program only receives data (see Section 3.13). Otherwise, the NTDI provides no particular advantage over an NTLA (Section 6.6.9) in terminating a logical link.

Up to 16 bytes of user data can be passed with the Disconnect Message.

BASIC–PLUS–2 Call Format:

CALL NTDI BY REF(*erval%,ula%,buffer?,buflen%,bufoff%*)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTDI call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The *ula* is an integer value giving the user link address (1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

buffer

This argument defines the starting address of a buffer containing the optional user message data. (See Section 6.1.1 for information on how to set up buffers in BASIC–PLUS–2.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer argument defines the number of bytes of user data to be sent: 0 to 16.

bufoff

This integer value defines an offset, from the beginning of the buffer specified by the *buffer* argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.

byte 1	20 | 21	(buflen+20)
	offset | user data	

whole buffer

NOTE

The BASIC–PLUS–2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. If, for example, a program's buffer is 20 bytes long, with an offset of 10 and a *buflen* of 16, 16 bytes will be sent: the last 10 bytes of the buffer, and 6 bytes of whatever is beyond the buffer in memory.

Possible Errors:

errval	**Error Text and Meaning**
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
4	NO ROOM FOR USER ON DEVICE There are outstanding unacknowledged messages on either the Data Message transmit queue or the Interrupt/Link Service transmit queue. All messages previously sent over the logical link must be acknowledged before the Disconnect Message can be sent. (If an immediate unconditional termination of the logical link is desired, use the NTLA call.)
5	CAN'T FIND FILE OR ACCOUNT The user link address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Disconnect Message over a logical link that has not yet been confirmed will cause this error.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The *ula* specified is zero. It must be within the range 1 to 255.

(continued on next page)

errval	**Error Text and Meaning**
19	DISK BLOCK IS INTERLOCKED The last segment sent over this logical link did not have the *eom* flag set. NSP will not permit the link to be disconnected unless all messages have been completely transmitted. (Note that this test for the *eom* flag is done regardless of the flow control option chosen by the remote program.)
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Disconnect Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example disconnects logical link 255 and includes user data explaining the reason for the disconnect. (The remote program would have to be coded to recognize this message and take appropriate action.)

```
\ ULA% = 255% &
\ ERRVAL% = 0% &
\ CALL NTDI BY REF(ERRVAL%,ULA%,"NOMORE",6%,0%)
```

6.6.9 Send Link Abort Message (NTLA) (DECnet)

The NTLA call is used to terminate a logical link immediately. If there are still messages in the transmit queues waiting for acknowledgment from the remote NSP, they are discarded. They may or may not have been received by the remote system (see Section 3.14).

Up to 16 bytes of user data can be sent with the message. If the NTLA is issued for an established logical link, the remote system is notified that the link has been broken and the user data is delivered to the remote program. If the NTLA is issued for a logical link awaiting confirmation from the remote program, the user data does not reach the remote program (see Section 3.14).

BASIC-PLUS-2 Call Format:

CALL NTLA BY REF(*erraval%,ula%,buffer?,buflen%,bufoff%*)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTLA call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The *ula* is an integer value giving the user link address (1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

buffer

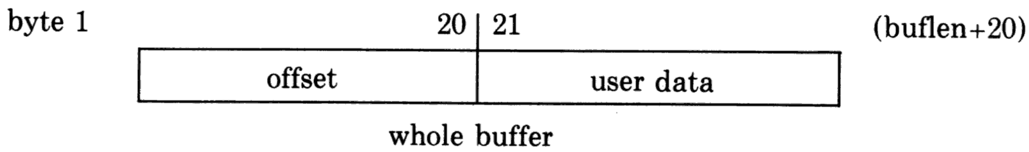
This argument defines the starting address of a buffer where the optional user message data begins. (See Section 6.1.1 for information on how to set up buffers in BASIC-PLUS-2.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by the buffer argument.

buflen

This integer value defines the number of bytes of user data to be sent: 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the address specified by the *buffer* argument. The value of *bufoff* is added to the starting address of the buffer to find the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



NOTE

The BASIC–PLUS–2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

Possible Errors:

errval	Error Text and Meaning
1	BAD DIRECTORY ON DEVICE Network operations have been terminated due to an internal error. Notify the system manager.
5	CAN'T FIND FILE OR ACCOUNT The user link address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
18	ILLEGAL SYS() USAGE One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The *ula* specified is zero. It must be within the range 1 to 255.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Link Abort Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later try may succeed.
62	NO RUN-TIME SYSTEM NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	MISSING SPECIAL FEATURE DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example aborts logical link 17. No user data is sent.

```
1000 MAP (BUFFER) A$ = 256% &
          .
          .
          .
     \ERRVAL% = 0% &
     \CALL NTLA BY REF(ERRVAL%,17%,A$,0%,0%)
```

6.7 Receive (MRCV) (Local and DECnet)

The MRCV call is used to retrieve a message from the calling program's pending message queue. It returns control parameters and user data, if any, to buffers defined in the call. The Connect Data Block for Connect Initiate Messages is also delivered to the buffer that receives the user data. The message data can be retrieved all at once or in portions, and portions can be discarded. The formats in Sections 6.7.1 – 6.7.9 show what control information is returned for the various types of messages.

A Receive call can be selected to get (1) the first message in the queue, (2) the first message in the queue from either a network or a local sender, or (3) the first message in the queue for a particular logical link. A "sleep" flag and associated timer can be set to indicate that the calling program is to wait if there are no appropriate messages pending in the queue.

BASIC–PLUS–2 Call Format:

CALL MRCV BY REF(*erval%,retbuf?,rmod%,sndr%,qual%,buffer?,*
buflen%,bufoff%,slptime%)

Argument Descriptions:

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MRCV call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the buffer to contain the control information identifying the type and characteristics of the message received. This argument must refer to a predefined buffer at least 40 bytes long. The MAP statement is particularly useful to define the buffer, as it both allocates the space for the returned information and allows the definition of named fields with different data types within the buffer (Section 6.1.1). The format of the data returned depends on the message type, as described in Sections 6.7.1 - 6.7.9.

rmod

The receive modifier (*rmod*) argument is an integer value consisting of four flags: *s% + t% + l% + n%*.

s The sleep flag (*s*) defines what is to be done if there is no appropriate message in the queue for the Receive call to deliver — that is, a message of the type requested with the *l*, *n*, *sndr*, and *qual* arguments. If $s = 1$, the program will "sleep," suspending execution until something happens. The *slptime* argument regulates the time that the program will sleep.

If the *s* flag is set to 0, the program will not sleep. If there are no appropriate messages in the queue the call terminates immediately with *errval* = 5.

t The truncate flag (t) is set to 2 to discard any excess information that will not fit in the buffer specified in this call. Information could be left over if the value of *buflen* is smaller than the amount of user data in the message.

A 0 value for t indicates that such excess information is to be kept for retrieval on later Receive calls.

l The local sender flag (l) is set to 4 to select a message from a local sender. The selection can be further qualified with the *sndr* and *qual* arguments.

If $l = 0$, local selection is not requested. If both l and n are zero, the first message in the queue is returned.

n The network sender flag (n) is set to 8 to select a message from a network sender. The selection can be further qualified with the *sndr* and *qual* arguments.

If $n = 0$, network selection is not requested. If both l and $n = 0$, the first message pending in the queue is returned. If $l = 4$ and $n = 8$, local selection prevails.

sndr

The sender selection argument (*sndr*) selects a particular local sender or a particular logical link for this Receive call.

If $l = 4$ (local selection) and *sndr* is nonzero, *sndr* is interpreted as a job number times two. The first message on the queue from that particular local job is retrieved.

If $l = 0$ and $n = 8$ (network selection), a nonzero value for *sndr* is interpreted as a user link address. The first message on the queue from that logical link is delivered.

If *sndr* = 0 when either $l = 4$ or $n = 8$, the *qual* parameter has special meaning. The *sndr* and *qual* arguments are ignored if both l and n are zero.

qual

The sender selection qualifier (*qual*) argument is normally zero for user applications.

If $l = 4$ (local selection) and *sndr* = 0, then any nonzero value for *qual* is a special-case Receive call requesting a message from the system (job number 0). This special case is used by the RSTS/E utility ERRCPY, which processes messages from the monitor error logging routines.

The *qual* argument is ignored if both l and $n = 0$, or if *sndr* $\neq 0$.

Table 6–4 summarizes the relationships between the selection bits of *rmod* and the *sndr* and *qual* arguments.

Table 6–4: Sender Selection Summary

rmod n	*rmod* l	*sndr*	*qual*	**Result**
0	0	n/a	n/a	The *sndr* and *qual* values are ignored. The MRCV call returns the first message in the pending message queue.
n/a	4	0	0	Selects the first Local Data Message in the queue.
		0	nonzero	Selects job 0. This combination is used by the error logging programs to select messages from monitor error logging routines.
		nonzero	n/a	Selects Local Data Messages by job number. The *sndr* argument is interpreted as a job number times 2. Only messages from that job are delivered on this MRCV call.
8	0	0	n/a	Selects the first network message (anything other than a Local Data Message).
		nonzero	n/a	Selects network messages from a particular logical link. The *sndr* argument is interpreted as a user link address. Only network messages from the designated logical link are delivered on this MRCV call.

buffer

This argument defines the starting address of the buffer to which user message data is to be delivered. (See Section 6.1.1 for a discussion of how to set up buffers in BASIC–PLUS–2.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by the buffer argument.

buflen

This integer argument defines the maximum number of bytes to be delivered to the buffer on this Receive call.

If a program has limited buffer space, the *buflen* value can be set to process a large segment in small pieces. If truncation is not requested (*t* flag of *rmod* = 0) and the amount of data in the message is greater than the amount indicated by *buflen*, then only the amount indicated by *buflen* will be returned and the rest will be saved for retrieval with later Receive calls.

The usual sequence in this case is to issue Receive calls until the number of bytes remaining in the pending message goes to zero. This point can be determined by examining bytes 9-10 in the data returned to the *retbuf* buffer by the Receive call. The number of bytes actually returned to the buffer is returned in bytes 13-14 in the *retbuf* buffer. (See the formats in Sections 6.7.1 – 6.7.9.)

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the buffer argument. The value of *bufoff* is added to the starting buffer address to define the beginning address of the user message data. For example, if *bufoff* = 20, the user message data is delivered beginning in byte 21 of the buffer.

byte 1	20	21	(buflen+20)
	offset	user data	

whole buffer

NOTE

> The BASIC–PLUS–2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. Up to 100 bytes beginning at *buffer*+20 could be delivered on the Receive call.

slptime

The *slptime* argument is significant only when the *s* flag of *rmod* = 1. In that case, a positive value for *slptime* defines the length of time, in seconds, that a program is to sleep when no appropriate message is in the queue. The program will sleep until:

1. The sleep timer expires.
2. *Any* new message is placed in the queue (not just an appropriate one).
3. A delimiter is typed on a terminal opened by, or assigned to, the job.
4. Log-ins are disabled. (This could occur if the system is being shut down).
5. A state change occurs on a pseudo-keyboard assigned to the job. (The job has printed output for the controlling job to read or has entered an input wait state.)
6. The job itself has opened the XM: (DMC11 driver) and a message is received for the job.

In all cases, the program is awakened with an error (*errval* = 5) but is not passed a message. To obtain a pending message the program must execute another Receive call. Since the program can be awakened by any of the conditions listed previously, a check for pending messages can be made by executing a MRCV without a sleep.

Possible Errors:

errval	**Error Text and Meaning**
5	CAN'T FIND FILE OR ACCOUNT For a Receive call without sleep (*s* flag in *rmod* = 0), this error indicates that no appropriate messages are pending. For a Receive call with sleep (*s* flag in *rmod* = 1), this error is returned when the program is awakened from the sleep. The program must execute another MRCV to retrieve any pending messages.
18	ILLEGAL SYS() USAGE No Receiver ID Block exists for the calling program. Before any Receive call can succeed, a Declare Receiver (MDCL) call must be executed.

Example:

In the following example, the first Receive call is issued for network messages, with the sleep bit set and the timer set to five seconds. The truncate bit is 0 and the length argument is 0, so all the information that is available is saved for the second Receive call. (The test for ERRVAL% = 5% after the first Receive call, with sleep, ensures that a message is waiting when the second Receive call is issued.)

```
1000 MAP (RET) RETBUF$ = 40%
     \MAP (BUF) BUFFER$ = 256% &
     \S% = 1% &
     \N% = 8% &
             .
             .
             .
     \ERRVAL% = 0% &
     \RMOD% = S%+N% &
     \SLPTIM% = 5% &
2000 \CALL MRCV BY REF(ERRVAL%,RETBUF$,RMOD%,0%,0%, &
       BUFFER$,0%,0%,SLPTIM%) &
     \IF ERRVAL% = 5% THEN GOTO 2000 &
     \RMOD% = N% &
     \CALL MRCV BY REF(ERRVAL%,RETBUF$,RMOD%,0%,0%, &
       BUFFER$,256%,0%,0%)
```

6.7.1 Format of Received Local Data Message

The format of the information returned to the *retbuf* buffer for a Local Data Message is shown below. Up to 512 bytes of user data can be delivered to the data buffer specified in the Receive (MRCV) call.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	-1%	The function code for a Local Data Message.
4	*jobx2%*	Two times the job number of the local sender.
5–6	*ppn%*	The project-programmer number of the local sender. Byte 5 = programmer number; byte 6 = project number.
7–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	The number of bytes of user data transferred to the buffer on this Receive call.
15–20	—	Not meaningful - should be ignored.
21–40	*param$*	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 20 bytes.

Sample MAP:

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,JOBX2$=1%,PROJ$=1%,PROG$=1%,FILL%, &
             REMAINDER%,FILL%,LGT%,FILL$=6%,PARAM$=20%
```

6.7.2 Format of Received Connect Initiate Message

The format of the information returned to the *retbuf* buffer for a received Connect Initiate Message is shown below. A 120-byte Connect Data Block and up to 16 bytes of user data are also returned to the data buffer specified in the Receive call. The format of a received Connect Data Block is shown in Figure 6–2 and described in detail in Table 6–5.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	-2	The function code for a Connect Initiate Message.
4	—	Not meaningful - should be ignored.
5–6	*lla%*	Local link address. Identifies the link for the local program's subsequent Connect Confirm or Connect Reject Message.
7–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	The number of bytes of user data transferred to the buffer on this Receive call.
15–22	—	Not meaningful - should be ignored.
23	*rlmod%*	Remote link modifier. Indicates the flow control in effect for the remote program. The low-order two bits should be masked and interpretted as follows: *flow%* = *rlmod%* AND 3% If *flow* = 0, no flow control = 1, segment flow control = 2, message flow control
24	—	Not meaningful - should be ignored.
25–26	*rmax%*	Receive maximum, in bytes. The maximum amount of user data that will be transmitted to the local program in one Network Data Message over this link. This value is supplied by the local NSP and is determined by the size of the receive buffers it allocates. The size of the receive buffers is set for NSP by Network Management. The local program can further limit the amount of user data to be received by specifying a smaller *rmax* in the Connect Confirm Message that accepts this request for a logical link (see Section 6.6.3).
27–28	*tmax%*	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the program in one Network Data Message over this link. This limit has been imposed by either the remote program or by the remote NSP. The local NSP enforces this limit for all Network Data Messages sent by the local program on this link.
29–40	—	Not meaningful - should be ignored.

Sample MAP:

```
MAP (RETBUF) A$ = 40
MAP (RETBUF) FILL%,CODE$=1%,FILL$=1%,LLA%,FILL%,REMAINDER%, &
             FILL%,LGT%,FILL$=8%,RLMOD$=1%,FILL$=1%,RMAX%,TMAX%
```

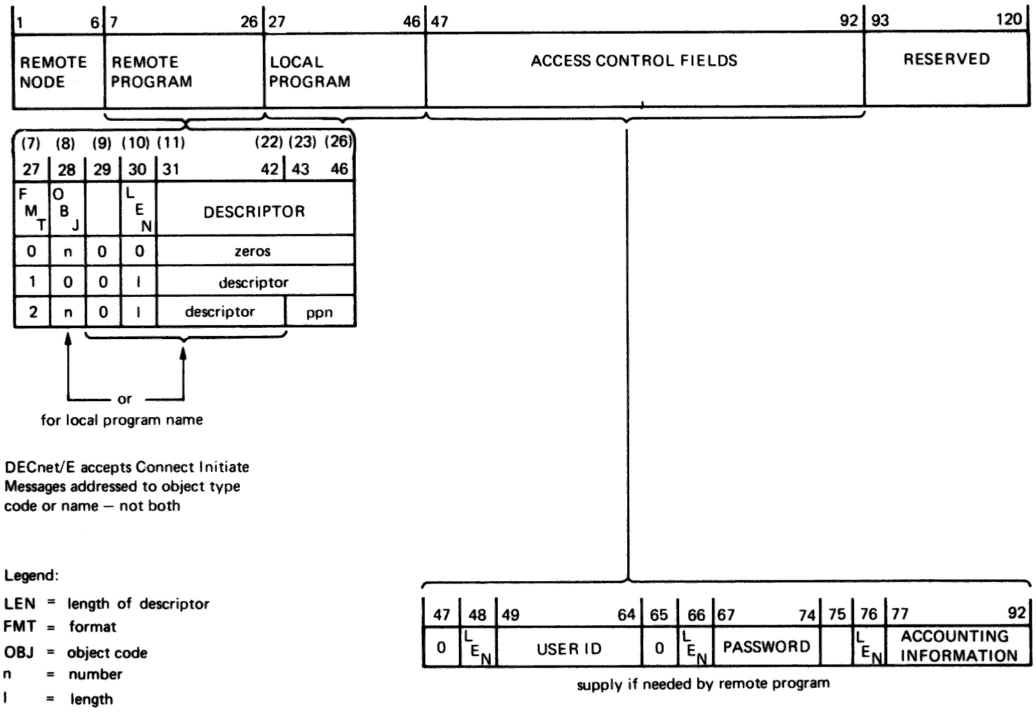


Figure 6–2: Format of Received Connect Data Block

Table 6–5: Format of Received Connect Data Block

Bytes **Content**

1–6 **Remote Node**. 1 to 6 uppercase, alphanumeric ASCII characters naming the remote node from which the Connect Initiate Message was received. The name will contain at least one alphabetic character. Node names shorter than six characters will be left-justified and padded to six characters with spaces.

7–26 **Remote Program**. Identifies the remote program that sent the Connect Initiate Message as either an object type (general function) or a specific entity (program, task, or process) This identification is given in one of three formats:

Format 0

The remote program is identified by object type code alone. The fields are:

Bytes	Content
7	Zero, to indicate a format 0 name.
8	Object type code of the remote program.
9–26	Not meaningful – should be ignored.

(continued on next page)

Table 6–5 (Cont.): Format of Received Connect Data Block

Format 1

The remote program is identified by a descriptor. The fields are:

Bytes	Content
7	One, to indicate a format 1 name.
8	Normally zero. This byte will usually be zero unless the remote DECnet system allows identification by both name and object type code on outgoing connect requests (DECnet/E does not). If nonzero, this byte is interpreted as the object type code of the remote program that sent the Connect Initiate Message.
9	Not meaningful - should be ignored.
10	Length of descriptor, 0 to 16 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11–26).
11–26	Remote program descriptor (left-justified). The descriptor used to register the remote program for network operations. For remote DECnet/E systems, this is the logical name declared in the Declare Receiver call.

Format 2

This format identifies the remote program by name or object type code and by the group and user codes under which the remote program is running on the remote system. (The group and user codes refer to what is called the project-programmer number on some DIGITAL systems, including RSTS/E. Other systems refer to these codes as the user identification code, or UIC.) DECnet/E systems use this format to identify the sender of a Connect Initiate Message.

The subfields are defined as follows:

Bytes	Content
7	Two, to indicate a format 2 name.
8	Object type code of the remote program, if applicable. If zero, the descriptor alone identifies the remote program.
9	Not meaningful - should be ignored.
10	Length of descriptor, 0 to 12 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11–22).
11–22	Remote program descriptor, if this is used instead of object type code. The descriptor will be left-justified in the field.
23–24	Remote group code. Binary value giving the group number under which the remote program is running. If the remote system is a DECnet/E system, byte 23 is the project number in a project-programmer number.
25–26	Remote user code. Binary value giving the user number under which the remote program is running. If the remote system is a DECnet/E system, byte 25 is the programmer number in a project-programmer number.

(continued on next page)

Table 6–5 (Cont.): Format of Received Connect Data Block

27–46	**Local Program.** This field indicates how the remote program addressed the local program in the Connect Initiate Message. DECnet/E allows the local program to be addressed by object type code or by name, but not both. Again, three formats can be used. The subfields are as follows:

Bytes	Content
27	Format type. Equal to 0, 1, or 2, depending on the format used by the remote program to address the local program.
28	Object type code, if used by the remote program to address the local program (formats 0 or 2).
29	Always zero.
30	Length of name, in bytes. Will be 0 for format 0, 1 to 6 for format 1, 0 to 6 for format 2. (DECnet/E program names are limited to a maximum of six characters by the Declare Receiver call.)
31-36	Local program name, if used by the remote program to address the local program (formats 1 or 2).
37-42	Always zero.
43-44	Local group code. Binary value giving the local program's project number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by NSP to identify the local program but passed on here.
45-46	Local user code. Binary value giving the local program's programmer number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by NSP to identify the local program but passed on here.

47–92	**Access Control Fields.** These bytes can be used to define the remote program's access rights to the local program's services. The idea is analogous to logging in to the local system. The most rigorous checking done in a normal log-in on DIGITAL systems involve user identification, password, and account number, so three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all nor does it require that the fields be used as described. A DECnet/E system simply passes this information on to the receiving program.

Bytes	Content
47	Not meaningful - should be ignored.
48	Length of user identification information specified in the next field, 0 to 16 bytes.
49–64	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the project-programmer number used at log-in.
65	Not meaningful - should be ignored.
66	Length of password information specified in the next field, 0 to 8 bytes.
67–74	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the project-programmer number at log-in.

(continued on next page)

Table 6–5 (Cont.): Format of Received Connect Data Block

	Bytes	Contents
	75	Not meaningful – should be ignored.
	76	Length of the accounting information specified in the following field, 0 to 16 bytes.
	77–92	Accounting information. Many systems require a billing account number in addition to user identification and password. The accounting field is provided for this purpose.
93–120		**Reserved for future use**. Should be ignored.

Sample MAP of Connect Data Block:

```
MAP (CDB) A$ e 120%
MAP (CDB) REMNODE$= 6%,REMFMT$=1%,REMOBJ$=1%, &
          FILL$=1%,REMLEN$=1%,REMNAME$=12%, &
          REMGRP$=1%,FILL$=1%,REMUSR$=1%,FILL$=1%, &
          LOCFMT$=1%,LOCOBJ$=1%,FILL$=1%, &
          LOCLEN$=1%,LOCNAME$=12%, &
          LOCGRP$=1%,FILL$=1%,LOCUSR$=1%,FILL$=1%, &
          FILL$=1%,IDLGT$=1%,USRID$=16%, &
          FILL$=1%,PWDLGT$=1%,PASSWRD$=8%, &
          FILL$=1%,ACCTLGT$=1%,ACCNT$=16%,FILL$=28%
```

6.7.3 Format of Received Connect Confirm Message

A received Connect Confirm Message is a remote program's acceptance of a Connect Initiate Message sent by the local program. The link is identified by the user link address specified by the local program in the Connect Initiate Message. The format of the information returned to the *retbuf* buffer is described below. Up to 16 bytes of user data can also be returned to the data buffer specified in the Receive call.

Bytes	**Value**	**Description**
1-2	—	Not meaningful - should be ignored.
3	-3%	The function code for a Connect Confirm Message.
4	*ula%*	User link address, previously defined by the local program in its Connect Initiate Message.
5-8	—	Not meaningful - should be ignored.
9-10	*remainder%*	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11-12	—	Not meaningful - should be ignored.
13-14	*length%*	The number of bytes of user data transferred to the data buffer on this Receive call.
15-22	—	Not meaningful - should be ignored.
23	*rlmod%*	Remote link modifier. Indicates the flow control in effect for the remote program. The low-order two bits should be masked and interpretted as follows: *flow%* = *rlmod%* AND 3% If *flow* = 0, no flow control = 1, segment flow control = 2, message flow control
24	—	Not meaningful - should be ignored.
25-26	*rmax%*	Receive maximum, in bytes. The maximum amount of user data that will be transmitted to the local program in one Network Data Message over this link. This will be (1) equal to the *rmax* value specified in the Connect Initiate Message for this link, or (2) a lesser value supplied by the local NSP, calculated from the size of the receive buffers it allocates.
27-28	*tmax%*	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or by the remote NSP. The local NSP enforces this limit for all Network Data Messages sent by the local program on this link.
29-40	—	Not meaningful - should be ignored.

Sample MAP:

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%,REMAINDER%,FILL%, &
             LGT%,FILL$=8%,RLMOD$=1%,FILL$=1%,RMAX%,TMAX%
```

6.7.4 Format of Received Connect Reject Message

A received Connect Reject Message is a rejection of the local program's request for a logical link. It can be from the remote program or the local or remote NSP. The link is identified by the user link address previously established in the local program's Connect Initiate Message. The format of the information returned to the *retbuf* buffer is described below. Rejections from NSP will contain a reason code explaining the reason for the rejection and will not contain user data. Rejections from the remote program can contain up to 16 bytes of user data, delivered to the data buffer defined in the Receive call.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	-3%	The function code for a Connect Confirm Message.
4	*ula%*	User link address, previously defined in the local program's Connect Initiate Message.
5–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	The number of bytes of user data transferred to the data buffer on this Receive call.
15–22	—	Not meaningful - should be ignored.
23–24	*reason%*	Identifies the reason for the rejection (see Appendix B).
25–40	—	Not meaningful - should be ignored.

Sample Map:

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%,REMAINDER%,FILL%, &
             LGT%,FILL$=8%,REASON%
```

6.7.5 Format of Received Network Data Message

A received Network Data Message contains user data from a remote program. The format of the information returned to the *retbuf* buffer is described below. The user data accompanying the Network Data Message is delivered to the data buffer specified in the Receive call.

Bytes	Value	Description
1–2	—	Not meaningful - should be ignored.
3	-5%	The function code for a Network Data Message.
4	*ula%*	User link address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	The number of bytes of user data transferred to the data buffer on this Receive call.
15–20	—	Not meaningful - should be ignored.
21	*dmflgs%*	Data message flags. Indicate whether this is the beginning, middle, end, or sole segment of a logical message. The low-order two bits should be masked and interpretted as follows: *bom%* = dmflgs% AND 1% *eom%* = dmflgs% AND 2% Then *eom* *bom* 0 0 Middle segment of message 0 1 First segment of message 2 0 Last segment of message 2 1 Sole segment of message DECnet/E uses the *eom* bit to determine the end of a logical message if the local program requested message flow control when the logical link was established. It does not use the *bom* bit but passes it on to the receiving program.
22–40	—	Not meaningful - should be ignored.

Sample MAP:

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%,REMAINDER%,FILL%, &
             LGT%,FILL$=6%,DMFLGS$=1%
```

6.7.6 Format of Received Interrupt Message

The local NSP places Interrupt Messages from a remote program at the head of the pending message queue, behind the first message and behind any other pending Interrupt Messages queued for the program. The format of the data returned to the *retbuf* buffer is described below. Up to 16 bytes of user data can be delivered to the data buffer specified in the Receive call.

Bytes	Value	Description
1-2	—	Not meaningful - should be ignored.
3	-6%	The function code for an Interrupt Message.
4	*ula%*	User link address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5–8	—	Not meaningful - should be ignored.
9–10	*remainder%*	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11–12	—	Not meaningful - should be ignored.
13–14	*length%*	The number of bytes of user data transferred to the buffer on this Receive call.
15–40	—	Not meaningful - should be ignored.

Sample MAP:

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%, &
             REMAINDER%,FILL%,LGT%
```

6.7.7 Format of Received Link Service Message

A Link Service Message provides status information for a particular logical link. A Link Service Message is only returned on a Receive call whenever the pending message queue has emptied after a condition has cleared that previously inhibited the local program from sending (see Section 4.2). Status information with the same format as a Link Service Message can also be returned to the *retbuf* buffer on successful completion of a Send Link Service, Send Network Data, or Send Interrupt call. The format of the information returned is described below. No data is delivered to the data buffer with a Link Service Message.

Bytes	**Value**	**Description**
1–2	—	Not meaningful - should be ignored.
3	-7%	The function code for a Link Service Message.
4	*ula%*	User link address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5–8	—	Not meaningful - should be ignored.
9–10	0%	Always zero for Link Service Messages. (These bytes are the remainder field in other returned messages.)
11–12	—	Not meaningful - should be ignored.
13–14	0%	Always zero for Link Service Messages. (These bytes are the length field in other returned messages.)
15–20	—	Not meaningful - should be ignored.
21	*lstat%*	Local status flags. These flags should be masked and interpreted as follows:

data% = *lstat%* AND 1%
intls% = *lstat%* AND 2%
lclbp% = *lstat%* AND 192% (2-bit field)

If *data* = 1, outgoing Network Data Messages are inhibited. (A Link Service Message will be delivered when the condition clears.)

If *intls* = 2, outgoing Interrupt and Link Service Messages are inhibited. (A Link Service Message will be delivered when the condition clears.)

The *lclbp* value indicates the local NSP's backpressure status:

- 0 Incoming Data Message flow is on.
- 64 Incoming Data Message flow is on but the local NSP will turn this link off if another Data Message is received before the local program's receive queue is emptied.
- 128 Incoming Data Message flow has been turned off by the local NSP. The local program's receive queue must be emptied before the link will be turned on.
- 192 Incoming Data Message flow has been turned off by the local NSP but flow is scheduled to turn on as soon as system buffers become available.

(continued on next page)

22	*rstat%*	Remote status flags. Only one bit is currently defined for this field. It should be masked and interpretted as follows: *rembp%* = *rstat%* AND 128% If *rembp* = 128, the link has been turned off by the remote system due to a backpressure condition. No Data Messages can be sent until the remote system turns the link back on.
23	*ldr%*	Local data request count. The count of segments or logical messages currently requested by the local program, reflecting the actual number of segments or logical messages requested but not yet received.
24	*lir%*	Local interrupt request count. The count of Interrupt Messages currently requested by the local program, reflecting the actual number of interrupts requested but not yet received. This count will never exceed one.
25	*rdr%*	Remote data request count. The count of segments or logical messages currently requested by the remote program, reflecting the actual count of segments or messages requested but not yet sent.
26	*rir%*	Remote interrupt request count. The count of Interrupt Messages currently requested by the remote end of the logical link, reflecting the actual count of interrupts requested but not yet sent.
27	*dtm%*	Data transmit queue maximum. The maximum number of data messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when NSP is enabled. It applies individually to all active logical links.
28	*dtc%*	Data transmit queue count. The number of data messages currently in the data transmit queue for this logical link. It is a count of data messages waiting for acknowledgment from the remote NSP.
29	*ltm%*	Interrupt/Link Service transmit queue maximum. The maximum number of Interrupt or Link Service Messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when NSP is enabled. It applies individually to all active logical links.
30	*ltc%*	Interrupt/Link Service transmit queue count. The number of Interrupt and Link Service Messages currently in the Interrupt/Link Service transmit queue for this logical link. It is a count of Interrupt and Link Service Messages waiting for acknowledgment from the remote NSP.
31	*mmax%*	Message maximum. The maximum number of incoming messages that will be queued for the local program. This maximum is set by the local program in its Declare Receiver call.
32	*mcnt%*	Message count. The total number of messages currently in the pending message queue for the local program.
33	*mula%*	Message count for this logical link. The number of messages currently in the pending message queue for the logical link identified by the *ula* in byte 4.
34–40	—	Not meaningful – should be ignored

Sample MAP:

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=16%, &
             LSTAT$=1%,RSTAT$=1%,LDR%=1%,LIR$=1%, &
             RDR$=1%,RIR$=1%,DTM$=1%,DTC$=1%, &
             LTM$=1%,LTC$=1%,MMAX$=1%, &
             MCNT$=1%,MULA$=1%
```

6.7.8 Format of Received Disconnect Message

A received Disconnect Message indicates that an established logical link with a remote program has been disconnected by the remote program. All other messages sent by the remote program over this logical link have been received by the local program. The format of the data returned to the *retbuf* buffer is defined below. Up to 16 bytes of user message data can be returned to the data buffer specified in the Receive call.

Byte	Value	Description
1–2	—	Not meaningful – should be ignored.
3	-8%	The function code for a Disconnect Message.
4	*ula%*	User link address, previously defined by the local program in its Connect Initiate or Connect Confirm Message.
5–8	—	Not meaningful – should be ignored.
9–10	*remainder%*	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11–12	—	Not meaningful – should be ignored.
13–14	*length%*	The number of bytes of user data transferred to the data buffer on this Receive call.
15–40	—	Not meaningful – should be ignored.

Sample MAP:

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%, &
             REMAINDER%,FILL%,LGT%
```

6.7.9 Format of Received Link Abort Message

A received Link Abort Message indicates that a logical link has been terminated by the remote program or by the local or remote NSP. The format of the data returned to the *retbuf* buffer is defined below. Up to 16 bytes of user message data can be returned to the data buffer specified in the Receive call.

Bytes	Value	Description
1–2	—	Not meaningful – should be ignored.
3	-9%	The function code for a Link Abort Message.
4	*ula%*	User link address (if any), previously defined by the local program in its Connect Initiate or Connect Confirm Message.
5–6	*lla%*	Local link address, established by the local NSP for this link. It identifies the link if the remote program sent a Connect Initiate Message but the local program has not yet responded with a Connect Confirm or Connect Reject Message. No *ula* exists under these conditions to identify the link being aborted.
7–8	—	Not meaningful – should be ignored.
9–10	*remainder%*	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11–12	—	Not meaningful – should be ignored.
13–14	*length%*	The number of bytes of user data transferred to the data buffer on this Receive call.
15–22	—	Not meaningful – should be ignored.
23–24	*reason%*	Abort reason. If nonzero, the link was aborted by either the local or remote NSP and the reason is specified in these two bytes. The reasons that apply for a link abort are listed in Appendix B. If reason is zero, the link was aborted at the request of the remote program.
25–40	—	Not meaningful – should be ignored.

Sample MAP:

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,LLA%,FILL%, &
             REMAINDER%,FILL%,LGT%,FILL$=8%,REASON%
```

Appendix A
Object Type Codes

Code	Type of Process
000	General Task, User Process
001	File Access (FAL/DAP Version 1)
002	Unit Record Services (URD)
003	Application Terminal Services (ATS)
004	Command Terminal Services (CTS)
005	RSX-11M Task Control, Version 1
006	Operator Services Interface
007	Node Resource Manager
008	IBM 3270-BSC Gateway
009	IBM 2780-BSC Gateway
010	IBM 3790-SDLC Gateway
011	TPS Application
012	RT-11 DIBOL Application
013	TOPS-20 Terminal Handler
014	TOPS-20 Remote Spooler
015	RSX-11M Task Control, Version 2
016	TLK Utility (LSN on DECnet/E)
017	File Access (FAL/DAP Version 4 and later)
018	RSX-11S Remote Task Loader
019	Network Management Listener (NICE Process)
020	RSTS/E Media Transfer Program (NETCPY)
021	Reserved for DECnet use
022	Mail Listener
023	Host Terminal Handler (NPKDVR)
024	Concentrator Terminal Handler

(continued on next page)

025	Loopback Mirror
026	Event Receiver
027	VAX/VMS Personal Message Utility
028	File Transfer Spooler (FTS)
029–062	Reserved for DECnet use
063	DECnet test tool (DTR)
064–127	Reserved for DECnet use
128–255	Reserved for customer extensions

Appendix B
NSP Reason Codes for Connect Reject and Link Abort

Code	Reason
000	No error – user-initiated reject or abort
001	Resource allocation failure
002	Destination node does not exist
003	Node shutting down
004	Destination program does not exist
005	Invalid destination name or source name field
006	Destination program's queue full
007	Unspecified error condition
008	Third party aborted logical link
009	User-initiated link abort
010–020	Reserved
021	Invalid destination address in Connect Initiate Message
022	Invalid destination address in Connect Confirm Message
023	Source address zero in Connect Initiate or Connect Confirm Message
024	Flow control violation – invalid request count in Link Service Message
025–031	Reserved
032	Too many connects to node
033	Too many connects to destination program
034	Access not permitted
035	Logical link services mismatch
036	Invalid accounting information

(continued on next page)

037	Segment size too small
038	User aborted, timed out, or canceled link
039	No path to node
040	Flow control failure - data received when request count zero
041	No current link (cannot recognize destination address)
042	Confirmation from remote system of Disconnect Message
043	Image data field too long

INDEX

M

N

O

P

Q

R

S

SET EXECUTOR, 4–10
SET OBJECT, 4–5
Sleep,
  in Declare Receiver, 5–6, 6–7
  in Receive, 3–3, 5–59, 6–52
Sleep monitor call, 5–6, 6–7
Source program, 2–2
Strings,
  in system calls (BASIC–PLUS), 5–2
  in system calls (BASIC–PLUS–2), 6–3
System buffer space, 2–8, 3–2, 3–4, 3–5, 5–7, 6–7
System calls,
  arguments (BASIC–PLUS), 5–2, 5–3
  arguments (BASIC–PLUS–2), 6–2 to 6–4
  general form (BASIC–PLUS), 5–1 to 5–2
  general form (BASIC–PLUS–2), 6–1 to 6–2
  summary, 3–1 to 3–2
System library, 6–4
System manager, 1–4, 1–5, 2–10, 4–1, 4–3, 4–4, 4–12, 4–7, 4–10, 5–14, 6–14

T

U

V

DECnet/E
NETwork Programming in
BASIC-PLUS and BASIC-PLUS-2
AA- H501B-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) __________

Name __________ Date __________

Organization __________

Street __________

City __________ State __________ Zip Code or Country __________

— — —Do Not Tear - Fold Here and Tape — — — — — — — — — — — — — — — — — —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE DOCUMENTATION
1925 ANDOVER STREET TW/E07
TEWKSBURY, MASSACHUSETTS 01876

— — — Do Not Tear - Fold Here and Tape — — — — — — — — — — — — — — — — —

Cut Along Dotted Line